Carlos Filipe Gouveia de Sousa

Licenciado em Ciências de
Engenharia Electrotécnica e de Computadores

Service-oriented Infrastructure to support the Control, Monitoring and Management of a Shop floor System

Dissertação para obtenção do Grau de Mestre em
Engenharia Electrotécnica e de Computadores

Orientador :	José António Barata de Oliveira, Professor Doutor, FCT-UNL
Co-orientador :	Gonçalo Moreira Cândido, Researcher, CTS-UNINOVA

	Júri:
Presidente:	Prof. Doutor João Paulo Branquinho Pimentão
Arguente:	Prof. Doutor Tiago Oliveira Machado de Figueiredo Cardoso
Vogal:	Prof. Doutor José António Barata de Oliveira

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Abril, 2014

Service-oriented Infrastructure to support the Control, Monitoring and Management of a Shop floor System

To my family

Acknowledgements

First of all, I would like to thank Professor José Barata for giving me the opportunity to develop the work presented in this thesis. Professor Barata also allowed me to have the chance to become an assistance lecturer in Faculdade de Ciências e Tecnologia which has enriched my knowledge and my experience as a professional.

A special thanks to my co-adviser Gonçalo Cândido. I first met him as my professor, but, as my academic road evolved and ended, he also became a good and cherished friend. Gonçalo spent a lot of time helping me with my work by sharing his knowledge in my thesis' subject.

To Ivan Delamer and Johannes Minor, the guys that I never met personally but helped with technical issues during my thesis work and provided me all the support that I needed.

To my academic friends whom I have met in my first college year: Luis Rodrigues, Catarina Tomás, João Moreira and Jorge Correia. Without their help it would not be possible to conclude my studies. The nights that we have spent on the University studying for the exams and finishing some works were a great experience of team work.

I also would like to mention my lab colleagues: Francisco Marques, André Lourenço, Ricardo Mendonça, Pedro Santana, Rogério Rosa, Luis Ribeiro, Pedro Gomes, Eduardo Pinto, Magno Guedes and Pedro Deusdado. They were the guys who invented a nickname for me: 'Douradinho'. Those guys were always on a good mood and those jokes on lab made my day much better.

I must not forget to mention one person which I also met in the lab, during my master studies: Giovanni Di Orio. He helped and taught me a lot, especially when he shared his experience in the automation field. He is the easiest person to work with: simple, professional and a big friend of mine.

To my long-life friends: Daniel Oliveira, Diogo Ferreira and Frederico Lopes who have been with me since the first years of my life. We have shared many happy moments that can never be forgotten.

A tremendous 'Thank You' to my family, especially to my parents. They gave me the opportunity to become what I am today. They always supported me in the good and bad

moments and allowed me to conclude my studies with success.

Last, but not least important, my biggest friend and girlfriend: Sofia. She is the one who hears all my complaints in the difficult moments and yet still supports and encourages me to do a better job. She is, by far, the best person that I ever met.

Abstract

Service-oriented Architecture (SOA) paradigm is becoming a broadly deployed standard for business and enterprise integration. It continuously spreads across the distinct layers of the enterprise organization and disparate domains of application, envisioning a unified communication solution. Service-oriented approaches are also entering the industrial automation domain in a top-down way. The recent application at device level has a direct impact on how industrial automation deployments will evolve. Similarly to other domains, the crescent ubiquity of smart devices is raising important lifecycle concerns related to device control, monitoring and management. From initial setup and deployment to system lifecycle monitoring and evolution, each device needs to be taken into account and to be easily reachable.

The current work includes the specification and development of a modular, adaptive and open infrastructure to support the control, monitoring and management of devices and services in an industrial automation environment, such as a shop floor system. A collection of tools and services to be comprised in this same infrastructure will also be researched and implemented. Moreover, the main implementation focuses on a SOA-based infrastructure comprising Semantic Web concepts to enhance the process of exchanging a device in an industrial automation environment. This is done by assisting (and even automate) this task supported by service and device semantic matching whenever a device has a problem. The infrastructure was implemented and tested in an educational shop floor setup composed by a set of distributed entities each one controlled by its own SOA-ready PLC. The performed tests revealed that the tasks of discovering and identifying new devices, as well as providing assistance when a device is down offered a valuable contribution and can increase the agility of the overall system when dealing with operation disruptions or modifications at device level.

Keywords: Service-oriented Architecture, Semantic assistance, Device management, Plug-and-play, Device Profile for Web Services

Resumo

O paradigma da Arquitectura Orientada a Serviços (SOA) é cada vez mais um standard amplamente difundido para a gestão e integração de empresas. Este paradigma estende-se continuamente pelos diferentes níveis da organização empresarial e pelos mais díspares domínios de aplicação, sempre orientado para uma solução de comunicação uniformizada. As abordagens orientadas a serviços também se inserem no domínio da automação industrial, numa abordagem "top-down". Ao nível do dispositivo, a recente aplicação tem um impacto directo na forma como a organização da automação industrial evoluirá. Tal como sucede em diversos outros domínios, a crescente ubiquidade de dispositivos inteligentes está a criar importantes preocupações ao nível dos seus ciclos da vida, relacionadas com o controlo, monitorização e gestão dos mesmos. Desde a montagem inicial até à monitorização e evolução do sistema de ciclo de vida, cada dispositivo necessita de ser tido em conta e de ser facilmente alcançado.

Este trabalho inclui a especificação e desenvolvimento de infra-estruturas modulares, adaptáveis e abertas, capazes de suportar o controlo, monitorização e gestão de dispositivos e serviços num ambiente de automação industrial, tal como é o caso de um sistema de linha de montagem. Uma colecção de ferramentas e serviços, que devem abranger a mesma infra-estrutura, foram também investigados e implementados. Concretamente, a principal implementação foca-se numa infra-estrutura baseada no SOA, abrangendo conceitos de *Semantic Web* para evidenciar o processo de troca de um dispositivo num ambiente de automação industrial. Tal é feito através da assistência (e até da automação) desta tarefa, suportada pelo serviço e pela correspondência semântica do dispositivo, sempre que exista algum problema. A infra-estrutura foi implementada e testada numa linha de montagem de ensaio, composta por uma série de entidades distribuídas, sendo certo que cada uma foi gerida e controlada pelo seu próprio *SOA-ready Programmable Logic Controller (PLC)*. Os testes levados a cabo revelaram que as tarefas de descoberta e

identificação de novos dispositivos, bem como a prestação de assistência quando o dispositivo se encontrava indisponível ofereceram uma valiosa contribuição, podendo aumentar a agilidade do sistema em geral, quando se lida com a interrupção da operação ou modificações ao nível do dispositivo.

Palavras-chave: Arquitectura orientada a Serviços, Correspondência Semântica, Gestão de dispositivos, Plug-and-play, Perfil de dispositivo para *Web Services*

Contents

List of Figures

List of Tables

Listings

1

Introduction

1.1 Motivation

Globalization is the integration and interdependency of world markets and resources in producing consumer goods and services, said [Koren, 2010]. Like so, it is fairly easy to conclude that globalization may be accountable for the evolution of the manufacturing industry, throughout the history, but mainly in the more recent decades.

Indeed, globalization offers new opportunities to an increasingly demanding society, giving industries the chance to offer and create new products which can be innovative and attract customers from all parts of the world.

Back when the craft production began its existence, the focus was on composing and creating products with the resources available at that time, in the most efficient possible way. But time went by and so did scientific discovers, e.g. steam engine, changing the production methods, causing different features in products to be more perfect and detailed.

A specific paradigm, common across this journey of evolution, was responsible for the effervescent development of several industry fields. The paradigm was mass production, associated to the models idealized by the well-known Henry Ford.

Industry continued evolving until today introducing new concepts to production lines such as flexibility and agility [Goldman et al., 1995] which led to research over Flexible Manufacturing Systems and later Evolvable Production Systems. These manufacturing systems are understood as enablers rapidly respond to changing markets and customer needs.

Interoperability tends to be considered simply as a technical issue and its real implications are sometimes underrated. As referred in [IEEE, 1990], interoperability consists in exchanging information between two systems and to use the information that

has been exchanged despite the differences in language, interface or execution platform. Nevertheless, it is more dependent from semantics more than over a simple software or hardware compability [Park and Ram, 2004]. Semantic interoperability emerged as the capability of automatically interpret the exchanged information precisely in systems [Heiler, 1995].

It is not sufficient to state that agility, alongside interoperability, corresponds to being flexible or lean since the concept extends more than that. Firstly, flexibility refers to an enterprise that can easily adapt itself to produce a range of planned products. Secondly but differently, lean essentially means producing without superfluous waste, reconfigurability denotes an ability to tackle uncertainty by relying on reconfigurable modules that can be composed to adapt to a new unpredictable situation.

Thus, an agile company must be able to adjust as quickly as possible in order to respond to the demands of globalization, environmental and working conditions regulation, improved standards for quality, fast technological mutation along with product variations in terms of form, functionality and throughput. Only if the lower level entity cannot accomplish the wanted agility target, will the overall system be incapable of delivering the desired performance. That is to say, in a complete system, its agility is always set back by its least agile element. Overall company agility is mainly supported by the device level in industrial automation, since a device is the last border where higher level process requirements, guidelines and workflows are turned into a structured collection of physical actions and services.

While these concepts were researched, two technologies emerged and were used to implement theories in manufacturing systems to increase the efficiency, e.g. flexibility and agility, of shop floor systems. Service Oriented Architecture and Multi Agent Systems are, until today, the responsible for trying to prove concepts thought before. Consequently, a fully implementation of the whole concepts described by recent manufacture paradigms was not achieved yet. As yet, a fully implementation of the whole concepts described by recent manufacture paradigms was not achieved.

Semantic Web is an envisaged Internet evolution in which the meaning of online information and services is clearly characterized and linked, making it possible for the web itself understand and satisfy the requests of people and machines to exploit it [Shadbolt et al., 2006]. Based on previous premises, the concept of semantic web services was born as complement to traditional web services technologies, as presented in [McIlraith et al., 2001]. Semantic web solutions such as ontologies and logic-based reasoning engines are considered agile factors to enhance enterprise integration tasks and can be extended to device level reengineering assistance and possible automation.

Nowadays, the systems integration community witnesses an increasing demand for new tools and services to assist its daily work and reduce the effort, cost and delay of lifecycle (re)engineering interventions. This is where the progressive evolution of the technology and availability of development platforms promote new options which may

become available to deploy innovative solutions that can support the deployment of features and functionalities previously unconceivable for the industrial automation.

1.2 Research Problem

The world of shop floor equipment is characterized by a high degree of diversity in device functionality, form factor, network protocols, input/output features, as well as the presence of many heterogeneous hardware and software platforms. In areas with a huge amount installed devices like automotive or aircraft industry, this origins a patchwork of technology islands known by its poor interoperability and scalability [Cândido et al., 2011].

Most expenses of a manufacturing plant during its lifecycle happen in its installation and setup, followed closely by maintenance downtime, in what concerns operating costs. A unified approach is therefore much needed to deal with the complete system lifecycle covering all its different phases of operation.

In this document a Service-Oriented Infrastructure is proposed to control, monitoring and manage a Shop floor System. Considering the fact that concepts like flexibility and agility are still looking for new answers, the nowadays systems proposed should be generic and easy to integrate to try to improve the current controls systems in order to increase its efficiency in the factories. To do this, a generic architecture has been developed and implemented in this dissertation divided in several components featuring a Device Explorer, responsible for analysis, setup and troubleshooting of a service-oriented application. Also a Semantic Assistant capable of presenting a reference test bench for a set of procedures and tools that allow a system integrator to perform a SOA-capable device exchange in a more effortless and natural manner by implicitly employing semantic web techniques and a transparent interoperability. In addition, a Process Management Tools was developed to define the set of processes to be executed by the system.

These type of infrastructures provide solutions on the most common issues presented in the production systems:

1. Excess time for replacing devices with problems during production.
2. Inflexible centralized implementations.
3. Shop floor isolated from higher level information.

Firstly, a State-of-the-Art Analysis will be presented describing the evolution of industry and a short explanation about the paradigms and techonologies that started since craft production until today. Then, Emerging Techonologies explaining the technologies which are capable to solve the theorical problems. After these analytical chapters an explanation about the infrastructure of this work is presented in the Architecture Overview followed by Implementation and Validation and in the end Conclusions and Future Work that can resume this work and allow or introduce new ideas for the future.

2

State-of-the-Art Analysis

In the early stages of civilization, materials were transformed and adapted by the Human Race to fulfil their survival needs. The existing resources, such as flint, and later, metals, were worked by Men's own hands, with rudimentary technic systems. With the advent of the progress of civilization, an increased specialization was verified, leading to the flourishing of a new group of workers, so called craftsman, who began to invent and create pottery, textiles, weapons, and other utility objects.

Craftsman business developed and by the end of Middle Age, they began to group into corporations so that they could be more proficient, passing and updating their knowledge. Inside each corporation, some were learners, others teachers and officers but together they only were able to allow a slow and low increase in productivity since the total course of the creation of the product, from start to finish, was made by the same craftsman [Jones et al., 1990].

It was by then that a new paradigm arrived: craft production. With this paradigm, production was performed by much skilled workers, using general-purpose machines, capable of creating exactly what the costumer requested. The down side was it could only make one product at a time, which of course, affected productivity [Piore, 1984].

Generally speaking, the industry evolution was distinguished by some events and crucial steps such as the invention of steam engine by James Watt, some researches with the water pump, steam powered from Thomas Newcomen and from Denis Papin, who studied elastic force form steam.

Specifically, the concept of industry is translated as the production of goods with the aid of machines. This concept had its first big boost in Great Britain, by the time of industrial revolution, in the last decades of the XVIII century. The industrial process in this era was characterized by a better production of iron by all machines. The industrial revolution quickly spread all over Europe and United States and allowed manufactured

products to be produced industrially[De Masi, 1999].

Later, during the First World War, there was, obviously, a great urge for weapons which required a high volume of production, impelling an update in all the industry's infrastructures.

Not long after, during the decade of twenties, industrialization continued to expand. Mechanization and the electrisation of factories permitted the augmentation of work productivity, mostly in Europe, United States and Japan and many sectors of activity were affected. From the organization and methods employed perspective, the work was systematized, especially in large assembly lines, first established in the automotive industry by the American Henry Ford introducing a new concept: mass production. This method, which is based on a moving assembling line, became the enabler of the mass production paradigm.

Mass production came to demonstrate that producing in a continuous-flow assembly line, while using interchangeable parts, with workers being assigned specific and systematized tasks would ultimately increase the overall quality of the product and allow a faster production and a significant price decrease. Note that this is the opposite from the mentioned craft production, where one-of-a-kind parts were individually made to fit together [Ford and Crowther, 1988].

As a product moves from station to station, it assembles parts in product that would fit and could be added without any alteration. This proves that interchangeable parts are the key of mass production [Koren, 2010].

Ford followed in a parallel process, with small teams of workers making multiple tasks on a single product, into a sequential process, where workers performed only minor tasks. The work was then transferred to the next team worker to perform the next small task over the product. This made sure the each worker's effort over that product was equitable [Taylor, 2002].

Overnight, this paradigm quickly became famous and well-succeed due to its low price per unit, though it was not customized to the customer's requests. As Ford once said: "You can have any colour you want, as long as it's black", statement which stands for the lack of diversity of products in this mass production paradigm. However, as previously mentioned, it increased the production at low cost, empowering a consequent reduction in the product price. That is, by lowering the price, a larger range of products became more affordable to the general public, which, in turn, increased the market of mass production factories.

Rigidity represented the main characteristic of shop floors during the existence of the mass production paradigm. The mission of shop floors was to produce as many units as possible at the lowest cost, generating an optimised production. By the mid-1980s, the mass production paradigm still remained, even so not concerning about any of the customer's preferences and needs, denoting that as product prices are lowered, more people would afford these products, increasing sales and market power.

Despite the temporary viability of this paradigm, mostly between 1950s and 1970s, it

ended up facing its substitute not long after, specifically in the early 1960s, so called lean production. The conversion began in the car industry, when Japanese manufacturers, led by Toyota, started to apply a more efficient system standard, eradicating all waste of work, time and material and detecting process errors easier and quickly. Consequently, the car price decreased. But, more relevant, the quality was better, in comparison to mass production [Womack, 1990].

Other relevant differences can be pointed, mostly related, as stated, with the production of high quality products, even at lower volumes and with the complete elimination of waste such as excess inventory, manpower and equipment.

These wastes may be grouped in eight different classes which are differentiated according to their main characteristics [Koren, 2010]. That is to say, waste can result from overproduction if a factory is producing more than is required by market need, consequently increasing inventory and labour cost. The waste can also occur with product defects since a defected product must be eliminated, sometimes with a complex process of extra activities in the system such as a launching delay on the other products. Excess inventories are also one probable cause of waste as long as they are maintained in factories. It may also be associated with the complicated transportation routes due to poor factory layout and unnecessary motion of parts of a product. Simple facts, such as waiting time when parts are held in buffers and machines before performing a task also produces waste. Avoiding waste must also concern with inefficient processing (poor distribution of tasks in each machine). Finally, waste can be related to people working in the factory who are under-utilized.

Evidently, as explained, lean manufacturing's main purpose is the absolute elimination of waste, but that also leads to the lowering of cost production and to an enhanced quality if the product, proving the mass production dogma wrong [Womack and Jones, 2003]: high quality and low cost in the same product is, indeed, possible.

Focusing now in the way in which the shop floor is organised in lean manufacturing, it is important to mention the improvement in workers skills: these workers are polyvalent and able to execute different tasks assigned to their team [Pine and Davis, 1999]. There is also a general sense of fulfilment in these workers. It is normal to feel more motivated when one's job does not consist in the same task over again. Lean manufacturing also invests in team responsibility and ownership of the team's assembly operation. Therefore, each work has quality checking responsibilities such as fixing a defect on the moment, even if it means stopping the whole line, making sure all products with defect are found every time.

However, even lean manufacturing had an alternative process, mainly adopted by Europe and the United States. It consisted in an investment in new and innovative technologies which possibilitated the development of optimization processes. The truth is that the development of microcomputers and its progressive introduction in industry also influence this technological evolution or shift. It also meant an increase in research

needs, especially those related with systems integration, because of the appearance of several and different computer systems.

But this era is also known for one of its main focuses: the creation of a global architecture capable of modelling the different tasks in a factory and providing an integrated view. At the same time, the computer integrated manufacturing (CIM) paradigm [Groover, 2007] developed as a valuable contribution to the increasing competitiveness of manufacturing factories, mainly by the introduction of automation and a higher need of using computers.

Because of this new paradigm, the word flexibility had a new, more relevant, meaning. To this day, flexible systems are one of the main features of manufacture systems. Studies are even made to research and develop new shop floors, as flexible and completely automated as possible. This proves that automation, combined with flexibility, is the solution to help and try to sort out the new objectives in manufacture.

However, reaching the main goal required the creation and development of concepts like flexible assembly systems (FAS) as well as flexible manufacturing system (FMS). Once they were established, it was easier for systems to be able to handle with different products and dissimilar demand from clients. Specifically, FMS consists in a reconfigurable set of workstations, interconnected by a flexible material handling system and its control is assured by an integrated computation system [Upton, 1992]. It allows the production of several products, much varied, on the same system, whereas its production capacity is much lower than that of dedicated lines and their initial capital cost is higher. Diversely, FAS is composed by assembly stations which are connected by an automated material-handling system. The invention of programmable industry robots had a key role here since it allowed an accelerated development of this system. Still, other differences pop out between FAS and FMS: the type of operations, machining for the FMS and assembly for the FAS. Note that FAS (fully automated) later was renamed as FAA (Flexible Automatic Assembly) systems.

Still, the complexity of the problem was much larger than simply a shift in the technology of the shop floor, investing substantially on the flexibility issue. A completely satisfying solution, which could fulfil the needs of globalization, was not found by these approaches since the market conditions were constantly changing. The 1980's marked the beginning of a new era: the mass customization popularized by Joseph Pine II [Pine and Davis, 1999]. The main distinction from the past paradigm is the variety of each product and the volume per product variant which is relatively small. The market dictations, where supply is much higher than demand provided the conditions for global competition and consequently consumers felt an opportunity to become more selective when purchasing products, fulfilling their needs with a bigger variety of options available in market.

Nonetheless, in mass customization, manufacturers continue to produce in quantity. Still, in the same family of products (a set of products that own core functions that are

common to all products), mass customization manages to maintain variety, at competitive costs which are similar to those established by mass production. Clearly, companies adaptability will be crucial in order to implement mass customization. Their production focus must be adapted and so must their vision about their target which is an unstable and unpredictable world with a different range of needs. As firstly mentioned by [Nagel and Dove, 1991], the problems presented in this paradigm create the notion of agile manufacturing. Also, FMS, previously mentioned, is impulsed by agile manufacturing. It is accepted that FMS can be utilized to produce a wide range of products as well as accommodate a few internal changes. However, the down side is that it can only allow working in a predictive environment. Here, agile manufacturing can assist, dealing better with uncontrollable matters such as uncertain environments [Maskell, 2001].

Not only is agility a concern, substantiated in a company-wide effort, contributing for its success by integrating all the areas but it also requires, in order to achieve that goal, some points to be validated [Vernadat, 1999, Barata and Camarinha-Matos, 2000, Barata et al., 2001, Leitão et al., 2001, Camarinha-Matos and Barata, 2002].

One of those points is related do political decisions since regulation is key in innovation and cooperation. Another one is customer focus. This point consists in an actual philosophy of the company on the customer, creating solutions which may increase the value of products or services sold to customers. The main objective is to attract the consumers attention so that they are not as aware in what concerns the cost of the product. Furthermore, there is the relevancy of information technology. This means that a close relationship to the customer depends of a good computation support. Technology can also originate flexible and agile shop-floors, which are needed to activate production systems. Then processes must be re-engineered, constantly redefined if that is the case or reorganized as a routine. The actual internal organization of the company must also be taken care of with the autonomy and higher education of workers, assigned in teams which cooperate with each other. Finally, there has to be a will to change. Everyone must be constantly aware of the surroundings, monitoring them and prepared to react whenever it is necessary.

Several approaches, capable of filling the agility requisites, can and must be mentioned: Bionic Manufacturing Systems (BMS) [Ueda, 1992], Holonic Manufacturing Systems (HMS) [Van Brussel et al., 1998, Babiceanu and Chen, 2006] and Reconfigurable Manufacturing Systems (RMS) [Koren et al., 1999, Mehrabi et al., 2000].

The functioning of natural organs inspires BMS. As it is commonly known, life forms are mainly supported by organs which consist of components such as clees. The BMS mechanism is similar or parallel with those biological features and suggests concepts for realizing essential properties of upcoming manufacturing systems [Tharumarajah, 1996].

At this point, the entire business world was being affected by the increase rate of changes. To cope with this situation, the chosen solution was HMS, which emerged in the Intelligent Manufacture Systems (IMS). The concept of holon in HMS consists in an autonomous and cooperative building block of a manufacturing system

[Babiceanu and Chen, 2006]. Not only is it composed by an information part but also, and often, by a physical processing part while it remains immerse in a holarchy (a group of holons where a holon can be part of another holon).

Soon enough, a new concept or manufacturing paradigm emerged as a consequence of the introduction of mass customization, correlated with the focus of FMS on it. This new paradigm was Reconfigurable Manufacturing Systems (RMS). Its main highlights are its ability for rapid adjustment production capacity and functionality, across a product family, which gets done by rearranging or changing its components [Koren, 2010]. Also, a key aspect is its capacity to be prepared for reconfiguration at all time, or to grow and change within the scope of its lifetime, and so it can respond to market fluctuations rapidly.

In the more recent days, the Evolvable Assembly Systems (EAS) [Onori et al., 2006] and Evolvable Production Systems (EPS) [Frei et al., 2007, Barata et al., 2007] have developed. Much alike BMS, HMS or RMS, these systems keep in mind the concept of equipment modularization and self-reconfiguration. The discrepancy is noted in the notion of intelligence: EAS/EPS can be set at a finer granularity level. They are composed by assorted reconfigurable task-specific and process oriented modules, allowing a continuous evolution hand to hand with the product and the correlative assembly process. As the parts are smaller, the less work entrusted to them so that it became easier to coordinate and structure the system. [Onori et al., 2006] reached the conclusion that EAS/EPS requires a precise set of qualitative features to be described. These features can be described, for instance, as module, consisting in any unit that can perform operations and integrate a specific interface and in which the level of emergence divers from granularity level. There is also granularity, which translates in the lowest level of device being considered within reference architecture. As the level of building block (tool, gripper) is lower, the emergent behaviour gets higher. System components may also be added or removed, in a feature named plugability. Available systems components may also be rearranged in order to perform new, but pre-defined, operations when a new module is added or to discard operations when an existing module is removed, as reconfigurability (interoperability) is also one of the features. And finally, evolvability, which enables new or redefined levels of functionality through the reconfiguration of the system.

It is quite certain that implementation is being confronted with some challenges or setbacks, which are blocking a fully implementation of the whole concepts described by recent manufacturer paradigms. And all this despite the development of the next generation of manufacturing systems, embodied by the setting of theoretical background by previous paradigms.

Nowadays, industrial automation has a typical control system [Zeeb et al., 2007] as it is shown in Fig. 2.1. It is divided into three main layers: the first one includes sensors and actuators, the second layer is control and the third is management. The process is monitored by sensors and collects data from the resources. The information of these resources is sent through a specific communication to the process control level and it is

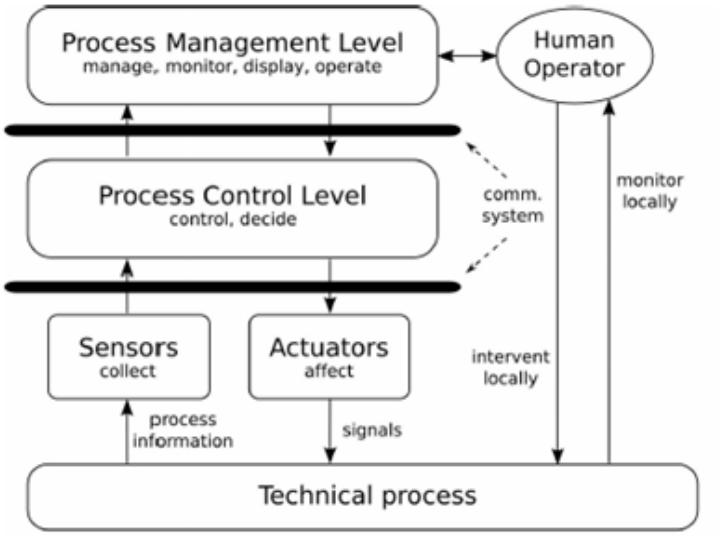


Figure 2.1: Control system

repeatedly evaluated by it. Based on this information, it decides which actuators need to be affected to interact with the technical process. At the same time, status information from the process control level is sent to the process management level. This is where human operators monitor the overall process behaviour, adjusting particular parameters and sending the configuration back to the process control level.

These type of systems are robust but at the same time they lack in terms of reconfigurability, flexibility and scalability. The robustness of these systems, along with the market-driven, consequence of mass customization, and the technology-pushed, explain the increasing number of intelligent systems. As a consequence and as a response, new research projects were created to try to respond to these needs, having in mind that Service-Oriented architectures could help solve these requirements from society. Reference projects such as SIRENA [1], ITEA SODA [2] or IST SOCRADES [3] were based on this idea.

SIRENA project emerged from an European Research and Development project aiming to develop a Service Infrastructure for Real time Embedded Networked Applications to support plug-and-play devices. The framework of SIRENA project provided the conditions for the implementation and production of Device Profile for Web Services (DPWS) stack, applied in a device level automation [Jammes and Smit, 2005a]. This project represented a split from traditional master-slave architectures and allowed a breakthrough in form of device networking. SIRENA project proved that there was still a need of creating new technologies for device integration in heterogeneous domains [Bohn et al., 2006]. As

[1] see http://www.sirena-itea.org/
[2] see http://www.soda-itea.org/
[3] see http://www.socrades.eu/

an achievement of the project, the first stack of DPWS was developed, giving the opportunity to other researchers to develop new ideas based on this concept, similarly to this work.

Later, SOCRADES (Service-Oriented Cross-layer Infrastructure for Distributed smart Embedded devices) was also an European project with a specific goal: develop a design, execution and management platform for next-generation industrial automation systems, exploiting SOA paradigm at the device and application level [Cannata et al., 2008]. This project showed the most relevant impacts that SOCRADES could have caused on the manufacture in terms of:

- Reconfigurability: SOA facilitates the discovery of devices, allowing the reconfiguration of each device instead of reprogramming the whole device.
- Interoperability: devices can be deployed in several types of platforms and networking technologies.
- Scalability: when there is a need of introduction of a new device in the production system, SOA allows it, avoiding a full reconfiguration of the system.
- Reusability: since every component can be organized and managed by any SOA-enabled system, the reutilization of each component can increase and consequently the reusability of the production system also increases.
- Maintenance optimization-Diagnosability: the opportunity to have a system capable of discovering and repairing failures in real-time.

Those impacts were used to prove, through the course of the project, that they can be applied in a production scenario through the theory and implementation of SOA concepts.

The SODA project follows the same guideline as SOCRADES project, focusing on the SOA framework, but addressing different application domains such as industrial and home automation, telecommunications and automotive electronics. The central focus on this project is the same as the others: development of factory automation devices capable of hosting DPWS enabled services, to improve the performance of the production system [de Deugd et al., 2006].

These three projects formed a robust background, providing several applications with DPWS implementation. It enhances the development of device communication, bringing new points of view to the manufacturing systems and some viable results to start small changes on it.

[Pohl et al., 2008] have presented a service-oriented control architecture for automation systems (Fig. 2.2).

This architecture forms a service hierarchy ranging from low-level sensor and actuator services, over a number of control service levels, up to application processes. This

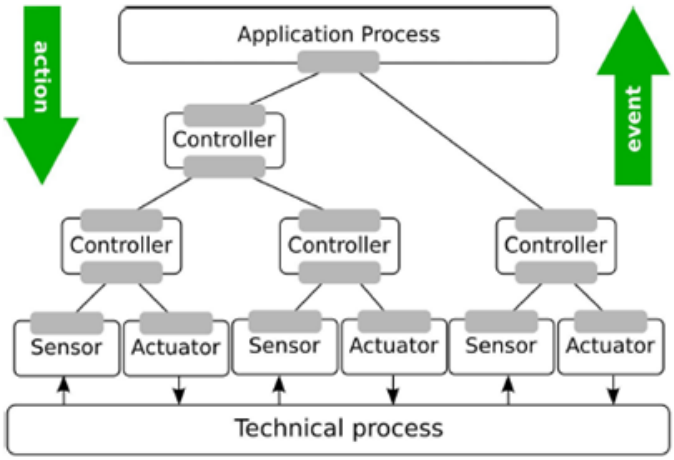


Figure 2.2: Control system

work was mainly focused on the evaluation of the functional behavioural of the control system allowing the checking of the current real-time limits of Java Virtual Machine and DPWS based control system implementations. It was proved in this work that this type of architecture it is not yet a feasible solution for all applications, particularly when dealing with high real-time requirements. [Cucinotta et al., 2009] also agreed that one of the big constraints of these architectures is the real time.

[Phaithoonbuathong et al., 2010] described in the present work that this types of architectures using DPWS as a way of communication between devices are obviously an advantage. The authors develop this idea by arguing that the key functionality of those architectures is in the provision of online discovery and service invocation for devices based on eXtensible Markup Language (XML) messages, passing among devices. [Zeeb et al., 2007] stated that one of the main pitfalls when implementing DPWS is the fact that clients only need, at maximum, three steps to access the description of a service that belongs to a device. The first step is the discovery of a device by the device type and the scope. The second step is the transfer of the device description and, later on, of the service description. This shows, in a way, the potential that DPWS can offer when having access to the service description: a detailed description of the operations which control the production systems, a standard communication between devices allowing the reusability and the introduction of new devices without reconfiguring the whole system.

[Marco et al., 2008] also suggest reconfigurable production systems build upon the concept of distributed control components that combine the features of SOA directed to automation and production systems. The usage of SOA in industrial automation provides a common ground for interoperability of all devices in a device network.

It is obvious that industry will not fully welcome these changes in the way of controlling and supervising the production systems just because research and development

projects tried to prove that they are capable of bringing advantages and features that can improve the production scenarios. Nevertheless, the main objective of these projects is to try to provide small accomplishments as well as solutions in order to try to introduce SOA technology in the scope of industrial automation. A proof of that result is the fact that there are some Programmable Logic Controllers (PLC) using services to display to the user the diagnostic about the hardware. The operator can then be ensured that if there is any fail on the hardware, he can know where the real problem is, avoiding wasting time to test every component.

To sum up, it is important to retain that these types of research and development projects have an important role on the development and evolvement of these matters in the future. Concepts and theories do not serve a merely academic purpose. They allow all end-users to develop their own critical thinking and analyse just in what extent it is worth to change to the described technologies, contrary to the versed ones. Nevertheless, one of the main deadlocks for the acceptance of the new technologies is the fact that the older ones actually work very well. It is now up to the researchers to present the advantages versus the old approaches to increase the acceptance of these new technologies.

3

Emerging Technologies

3.1 Service-oriented Architecture

The next evolutionary step to help IT organizations meet their ever more-complex challenges might just be Service Oriented Architectures. This technology as for main purpose the ability to solve the deficiency of consistent architectural framework that allows the speed of development, integration and the reuse of applications [Channabasavaiah et al., 2003].

[Jammes and Smit, 2005b] defines SOA as "a set of architectural tenets for building autonomous yet interoperable systems." The attention from the information technology scientific community towards the SOA paradigm has increased a lot due to the advent of new technologies such as web services, which allow the SOA paradigm to be perceived as a promising approach to create platform agnostic interoperable systems.

Theoretically, SOA has the potential to offer the needed system-wide visibility and device interoperability in complex collaborative automation systems subject to frequent changes. Furthermore and in practice, SOA constructs applications out of spread software services. They can implement, for instance, a transaction over the internet, a functionality typically recognized by most humans as a service. Their functioning consists of using defined protocols, describing and defining the interaction between two or more services, contrary to services embedding calls to each other in their source code.

There are three main properties of a service in SOA, which is an exposed piece of functionality [Hashimi, 2003]. These properties consist in the ability of the interface contract to the service to be platform independent, the possibility for the service to be dynamically located and invoked and the self-contain of the service.

The first main property means that any client from any Operating System has the possibility to consume the service chosen. Through dynamic discovery, it is suggested

that a discovery service is accessible. Consumers, or users, may find the wanted service using a look-up mechanism enabled by the directory service. That is to say, if a user wants to search for a credit card authorization service, he will question the directory service for available services which can provide and authorize a fee for a credit card. The service will then be selected according to the fee.

Note that exchanged messages between service providers and consumers are also a meaningful aspect of SOA. What happens is that those services expose an interface contract, not only capable of defining the behaviour of the service but also of doing it for the messages accepted and returned. The core idea of this architecture is to assure the integration between different platforms, as previously mentioned. Consequently, the referred technology has to maintain itself agnostic to any upcoming specific platforms. This is why messages are typically created using XML that conforms to XML schema.

Despite the concepts behind SOA, Web Services technologies are more likely to be assumed in it as the concerning fact standard to deliver effective, reliable, scalable and extensible machine-to-machine interaction. This also contrasts with their predecessors (CORBA and Distributed Computing Environment (DCE)).

However, the trigger for Web Services technology to emerge were technology demands and cultural prerequisites introduced by [Channabasavaiah et al., 2003]. There are some specific features or characteristics to these demands and pre-requirements, such as ubiquitous, open-standards-based, low-cost network infrastructures and technologies. These infrastructures and technologies are capable of offering a distributed environment much more conducive to the adoption of Web services than both CORBA and DCE.

Others can be referred, such as the level of acceptance and technological maturity to operate within a network centric environment, requiring interoperability to reach critical business objectives, such as distributed collaboration. Also, there is a generalized consensus that low-cost interoperability is best achieved through open Internet-based standards and related technologies.

The maturity of network-based technologies, e.g. TCP/IP tools sets platforms and connected methodologies that provide the infrastructure needed to ease loosely-coupled and interoperable machine-to-machine interactions.

3.1.1 Web Services overview

"A Web service is an interface that describes a collection of operations that are network accessible through standardized XML messaging" [Kreger et al., 2001]. The description of a Web Service is done through a standard, format XML notion, known as service description. It includes a certain amount of information, which is fundamental for association and interaction with the service, including message formats (which detail the operations), transport protocols and location.

The main features and abilities of the interface are its total independence from the

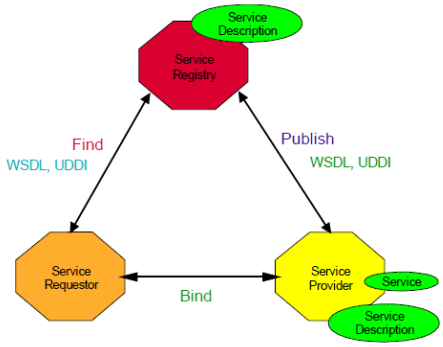


Figure 3.1: Web Services roles, operations and artifacts [Kreger et al., 2001]

programming language in which it is written, the concealing of the implementation details of service and the enabling of the interaction between different operating systems. These features and abilities instigate Web Services-based applications to be loosely coupled, component-oriented and cross-technology implemented. Web Services can be used alone or with other Web Services to implement a complex aggregation.

3.1.1.1 Web Service Model and Architecture

There are basically three main roles interacting in Web Services architecture. As it can be perceived in Fig. 3.1, these interactions include publishing, finding and binding operations.

The first main role, service provider, is perceived as the owner of the service. From an architectural viewpoint, the function of this platform is to host access to the service. Differently, from a business perspective, service requestor is the one which requires the satisfaction of certain functions.

Returning to the architectural perspective, this application may be characterized as the searcher of a service, so that it can invoke and start an interaction with it. Also, there is the role of service registry, which has service descriptions capable of being searched and found, as well as accepting publishing of service description by service providers. Lastly, the role of service is to find services and obtain binding information for services throughout the development of static binding or the execution for dynamic binding.

Publishing, finding and binding are the three main sorts of operations which constitute the Web services architecture. The first one, publishing, is meant to allow the availability of service description so that the service requestor is able to find a service. Secondly, the find operation, in which the service requestor retrieves a service description. And third, binding, which consists in allowing an interaction between the service requestor and the service chosen. To be correctly invoked, binding details can be found in service description.

Fig. 3.1 shows two artifacts which must be referred. On one hand there is service,

which is defined by a software module deployed on the network accessible platforms furnished by the service provider. On the other hand there is service description, which contains all the specifics of the interface and implementation of the service.

Then again, the division of the Web Services framework is done in three areas, known as communication protocols, service descriptions and service discovery defined by specifications [Curbera et al., 2002].

3.1.1.2 SOAP Communication

Consisting in a lightweight protocol for exchange of information in a decentralized, distributed environment, Simple Object Access Protocol (SOAP) is a three part XML based protocol. The respective three parts are, firstly, an envelope capable of defining a framework which subsequently describes the content of a message and the way in which it should be processed. The second part consists in a set of encoding rules sorted out to express instances of application-defined datatypes. And finally, the third part is a convention for representing remote procedure calls (RPC) and correspondent responses [Box et al., 2000]. Instead of defining an original protocol, SOAP was built to work on an already existing transport protocol, such as HTTP, SMTP or MQSeries.

SOAP messages are organized with a very simple structure: an XML element with two child elements, one of which contains the header and the other the body. The example of Listing 3.1 shows a SOAP envelope's structure.

Listing 3.1: Structure of a SOAP message

```
1 <SOAP:Envelope xmlns:SOAP=
2 "http://schemas.xmlsoap.org/soap/envelope/">
3 <SOAP:Header>
4 <!- content of header goes here ->
5 </SOAP:Header>
6 <SOAP:Body>
7 <!- content of body goes here ->
8 </SOAP:Body>
9 </SOAP:Envelope>
```

In SOAP messages there is a sender and a receiver between whom one way transmissions take place. Meanwhile, the combination of SOAP messages occurs in order to implement patterns such as request/response. They are routed along a path, better known as "message path", allowing the processing at one or more intermediate nodes, in addition to the ultimate destination. Note that this happens whatever the protocol to which SOAP is bound.

As previously mentioned, basic communication is offered by web services, with the aid of SOAP, though it does not mention which messages have to be exchanged so that the interaction with the service may be successful. Consequently, IBM and Microsoft then developed a web service description language (WSDL) capable of filling up and solving this problem. "WSDL is an XML format for describing network services as a set of endpoints

operating on messages containing either document-oriented or procedure-oriented information" [Christensen et al., 2001]. More than a set of definitions, a WSDL document has six min elements characterizing it. The first one, is types since WSDL documents provide data type definitions which are used to describe exchanged messages. The messages, in turn, are a representation of an abstract definition of the data transmitted at that time. They consist of logical parts and each on is relation with a definition within some type system. Another element is *portType*, representing a set of abstract operations and messages involved. Correlated to *portType* is *Binding*, which specifies a concrete protocol and data format specifications for the operations and messages defined by a particular *portType*. *Port*, in turn, has the ability to specify an address for a *binding*, defining, in that way, a single communication endpoint. And finally, there is *service*, mainly used to aggregate a set of related ports.

It is crucial to have some type of agreement on a vocabulary so that communication and interaction may be facilitated. This is exactly what WSDL is capable of enabling, providing a formalized description of client-service interaction. The most common service used by most services is XSD, considered "universal" for information exchange purpose. With that being said, WSDL can nevertheless support any type system.

Universal Description Discovery and Integration (UDDI) "is the definition of a set of services supporting the description and discovery of business, organizations, and other Web services providers, the Web services they make available, and the technical interfaces which may be used to access those services" [Bellwood et al., 2002]. It is characterized by two main specifications which define a service registry's structure and operation. On one hand there is the definition of the information and its way of encode, for each individual service. On the other hand there is a query and update Application Programming Interface (API) for the registry capable of describing the way in which the information can be accessed and updated.

3.2 Multi-Agent Systems

Automated manufacturing systems have witnessed an important new approach known as Multi Agent Systems (MAS), which is formed by a group of agents interacting and communicating in the same network.

"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives" [Wooldridge, 2008] with a common set of features widely accepted [Camarinha-Matos and Vieira, 1999]. These features include, for instance, autonomy, when, independently from third parties, an agent can operate alone and social ability, meaning that agents are able to interact/communicate with other agents or even other entities. Agents are also known for their reactivity when confronted with changes perceived in their environments and pro-activeness, according to the level of control of its own or the level of initiative during their operation. Agents have also mobility since

they have the ability to move through a net of execution environments. Their operation is continuous: this means that their running processes are continuous instead of one shot computation units. Also note their adaptability since they are able to change their behaviour depending on the environment perceived.

It can be stated that MAS revolves around the interactivity among agents, which enables them to not only solve problems but also accomplish objectives in a way that an individual agent would find much harder. It is "a group of agents organized according to specific, precisely defined principles of community organization and operation (architecture, messaging style, negotiation protocol, and so on) and supported by an adequate agent platform or infrastructure (registration, deregistration, communications support, and so on)" [Marik and McFarlane, 2005].

In the MAS context it is certain that the various agents decisions and actions have to necessarily interact. Even though this interaction cannot be questioned, it is not sufficient to solve all the problems presented by the MAS surrounding environment. Through internal mechanisms of the components of the agents the resolutions of all problems is not guaranteed [Monostori et al., 2006]. With that said, it is crucial for an agent do coordinate its actions with others, coming from other agents. This is important for the agent to decide according to friendly agents actions.

3.3 Service-oriented Architecture versus Multi-Agent Systems

The previous two chapters presented the two technologies which have potential to solve manufacturing paradigms so that new solutions may be accomplished. However, it did not present a comparison about SOA and MAS. SOA and MAS must be perceived as complementary technologies and not as concurrent tools. Furthermore, a comparative analysis between them can be found in Fig. 3.2.

The author also refers the main differences between these two technologies when tested in a production scenario: SOA emphasizes contract-based descriptions of the hosted services and does not provide a reference programming model. This happens even though both these paradigms contain the concept of distribution of autonomous entities, providing an effective modeling metaphor for complexity encapsulation. Differently, MAS supports well established methods to describe the behavior of an agent. Heterogeneity is one of the main features of automation environments. This lack of a structured development model/template renders system designing, implementation and debugging harder. For instance, this fact is crucial, in the case of the production paradigms earlier referred, since it consists of a system which undergoes dynamic runtime changes. Again, differently, SOA is usually supported by widely used web technologies and assures interoperability with a wide range of systems and can easily spawn over the internet. This is the opposite of what happens with MAS platforms, which are optimized for LAN use and are restricted to compliance with well defined but less used interoperability standards.

Characteristics	SOA	MAS
Basic Unit	Service	Agent
Autonomy	Both entities denote autonomy as the functionality provided is self-contained	
Behaviour description	In SOA the focus is on detailing the public interface rather than describing execution details.	There are well established methods to describe the behaviour of an agent.
Social ability	Social ability is not defined for SOA nevertheless the use of a service implies the acceptance of the rules defined in the interface description	The agents denote social ability regulated by internal or environmental rules
Complexity encapsulation	Again, the self-contained nature of the functionalities provided allows hiding the details. In SOA this encapsulation is explicit.	
Communication infrastructure	SOA are supported by Web related technologies and can seamlessly run on the internet.	Most implementations are optimized for LAN use.
Support for dynamically reconfigurable run-time architectures	Reconfiguration often requires reprogramming	The adaptable nature of agents makes them reactive to changes in the environment.
Interoperability	Assured by the use of general purpose web technologies.	Heavily dependent on compliance with FIPA-like standards.
Computational requirements	Lightweight implementations like the DPWS guarantee high performance without interoperability constraints	Most implementations have heavy computational requirements

Figure 3.2: Comparative Analysis between SOA and MAS [Ribeiro et al., 2008]

3.4 SOA at a device level: Device Profile for Web Services

Over the years, so-called middleware have been developed to implement different points of view of SOA. This approach to device-level SOA has already been adopted several years ago by the Universal Plug and Play[1] (UPnP) and JINI[2].

A series of technologies such as IP, TCP, UDP, HTTP, SOAP are used by UPnP to enable communication between devices. It may be described as a truly platform-agnostic. Nonetheless, it uses specific protocols for device discovery and eventing, and a specific XML language for service and device description. Differently, since it is a JAVA based solution, there is JINI, which provides mechanisms for discovering services and therefore lacks platform-neutrality in devices, forcing the existence of a Java Virtual Machine in every device. Then again, a very promising approach appeared, known as Devices Profile for Web Services, containing equal advantages to UPnP, but adding its fully integration with Web Services technology.

The DPWS consists in a plug-n-play protocol middleware built on top of a set of Web Services specifications and tackles description, discovery and control of services and devices on local networks. Two types of services are run by devices in DPWS architecture: hosting services and hosted services. There is a main difference between hosting services and hosted services: the first ones are directly associated to a device which plays an important role in the discovery process and the second ones are mainly functional, depending on their hosting device for discovery.

Furthermore, DPWS is also able to specify a set of built-in services [Jammes et al., 2005] such as discovery services, metadata exchange services and publish/subscribe eventing services. Discovery services are used by a service, which is connected to a network, allowing its own advertising and discovering of other devices. Metadata exchange services are able to provide dynamic access to a device's hosted services and to their metadata, for instance, WSDL or XML Schema definitions. Finally, Publish/subscribe eventing services are capable of allowing other devices to subscribe to asynchronous event messages produced by a given service.

As seen below, DPWS is based, only partially, on the Web Services Architecture (WSA), using further standards from the Web services protocol family. These standards are, for instance, WSDL, used in what concerns the abstract description of services interfaces and their binding to transport protocols. There is also the XML Schema for the definition of the data formats used for constructing the messages addressed to and received from services and SOAP which is the protocol responsible for transporting service-related messages according to WSDL standards.

[1] see http://www.upnp.org
[2] see http://www.jini.org

3.4.1 WS-Addressing

Critical messaging properties have to be carried across multiple transports. This requires a common mechanism so that messages may be routed and addressed in the mentioned multi-transport world. Therefore and for this purpose, there are three sets of SOAP header blocks which are defined by addressing specification, so called *Action*, *MessageID* and *RelatesTo* and *To/ReplyTo/FaultTo* headers blocks.

In the particular case of *MessageID* and *RelatesTo* headers blocks, messages can be uniquely identified through simple URIs. Differently, *Action* header block is capable of previewing the expected processing of a message. Finally, *To/ReplyTo/FaultTo* header blocks are the indicated ones to identify the responsible agents which are tasked to process the message and its replies. As a consequence of all these headers coming together as one, a new endpoint reference (WS-Addressing-defined structure) emerged. Its skills include the ability to bundle all the information so it can be properly addressed in a SOAP message.

3.4.2 WS-Policy

WS-Policy provides the needed mechanisms to allow Web services applications to specify policy information. This specification can be defined not only as an XML Infoset named a policy expression which contains domain-specific and Web service policy information but also as a core set of constructs created to indicate the way in which choices and/or combination of domain specific policy assertions are applied in a Web services context [Bajaj et al., 2006].

But the utility of WS-Policy does not end here since it can be used to express policies which are related to a Web Service in the form of policy assertions. Also, it accompanies the respective WSDL of the service.

3.4.3 WS-MetadataExchange

When an endpoint is associated to retrieve metadata there is a specification, which concerns the definition of data types and operations [Ballinger et al., 2004]. What other endpoints need to know to interact and communicate with the selected endpoint can be provided by this metadata, since in contains in itself the necessary information to do so.

3.4.4 WS-Security

This specification describes enhancements to SOAP messaging to provide message integrity and confidentiality [Atkinson et al., 2002]. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies.

3.4.5 WS-Discovery

The main feature of WS-Discovery is the ability to define a multicast discovery protocol, capable of locating services [Roe et al., 2005]. There is a primary mode of discovery, consisting of a client searching for one or more specific services. Three different endpoint types are introduced by discovery, so called target service, client and discovery proxy.

In the case of target services, they consist of Web Services offering themselves and being available to the network. The most wanted target services are those which can be discovered dynamically. And these are the ones which capture the clients interest. Differently, discovery proxy is an endpoint charged of controlling the services discovery over the network.

The client interacts with these services, using the *Probe* messages as an attempt to discover target services in the network. The act of matching target services will answer with the Probe Matcher messages send as UDP unicast message to the client.

Even though this is a valid discovery, at times, when a target service goes online, it may send a *Hello* message as UDP multicast and a client can listen for it and realize where the target service is located through that message, therefore understanding, also through that message, where the target service is. For the logical addresses introduced by the endpoint structure in *WS-Adressing* to be resolved, the client selects the *Resolve* message. Subsequently, target service will responde with the *Resolve Match* message send as UDP unicast to the client. Thus, whenever a service goes offline, a multicast *Bye* message is sent.

3.4.6 WS-Eventing

As events occur throughout other services and applications, web services often want to receive messages. Consequently, there occurs a need for a mechanism to register interest because of the set of Web services interested in receiving such messages, which is often unknown in advance or will change over time.

WS-Eventing is in charge of defining a protocol for managing subscriptions. For one Web service, exists a subscriber, to register interest, a subscription, with another Web service, an event source, in receiving messages about events and notifications [Curbera et al., 2004]. The subscription is leased by an event to an event sink and expires over time. Three built-in operations are provided by *WS-Eventing*, so called subscribe, renew and unsubscribe.

3.4.7 DPWS protocol stack

In conclusion, the Web Services protocol suit has suffered and extension with a specifically targeted profile at a device space: the "Device Profile for Web Services". The next major upgrade of UPnP will probably be foreshadowed by DPWS specification. DPWS protocol stack is shown in Fig. 3.3.

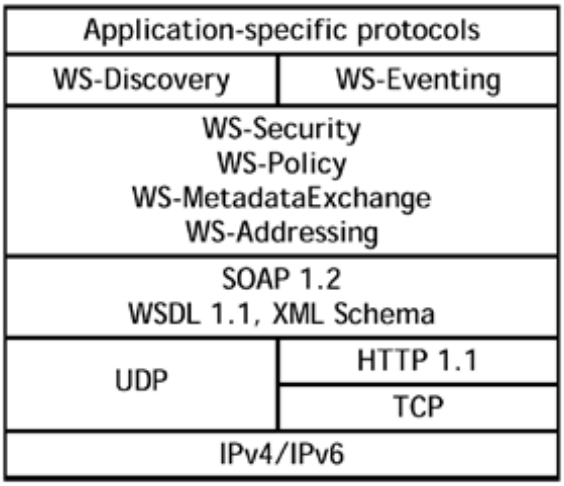


Figure 3.3: Devices Profile for Web Services protocol stack

3.5 Ontologies

The ontology concept can be described as a representation vocabulary, often specialized to some domain or subject matter [Chandrasekaran et al., 1999]. Different types of elements can be designated by ontologies, especially in engineering. These elements may include vocabulary describing conceptual elements and the relationship between them.

Sometimes, two systems have the need to be interoperable between them and that is where ontologies play their main function, providing and creating that much needed interoperability. The concept of interoperability can be defined as the ability of two or more systems or components to trade and subsequently use the exchanged information, despite their differences in interface, language and execution platform [IEEE, 1990].

Nevertheless, the independence found in interoperability is remarkable in what concerns content semantics, and even more so than over simple software or hardware compatibility [Park and Ram, 2004]. Semantic interoperability first appeared as the automatic acquisition on interpretation of the exchanges information in involved systems or components [Heiler, 1995].

It is mandatory, in this context, that the exchanged information is correctly and unambiguously defined. In other words, the perception of the transmitter must match to the understanding of the receiver. Thus, the focus was also on the way in which devices can be semantically described through ontologies so that not only the main differences between same services may be understood, but also the recognition of which device is semantically equal to other is reachable.

Ontologies are increasingly being recognized as an important matter and this is due to the augmentation of use of computer agents. Since software agents are becoming more independent from humans, there is a need for a common language that allows agents to understand each other while communicating. It is also due to the increased knowledge of

management practices because of the need of organizations to structure and maintain information creating a common information definition seen as a very valuable asset. Also, the importance of the World Wide Web for organizations and personal users must not be forgotten, and it recently has led to the creation of the concept of Semantic Web, providing means to comprehend machine data as the opposite concept which is the current one, where all data is only human understandable [Oliveira, 2003].

There is no doubt that ontologies create high expectations in the future, as to playing an important role in aiding automated processed and intelligent agents to access and interpret information. Particularly, ontologies will be presumably used to provide structured vocabularies, able to clarify the relationships between different terms. This ability will allow machines to communicate among them, regardless of human interpretation.

When the meaning of a term is well-defined in ontologies, then it can be used in a semantic markup, able to describe the content and functionality of Web-accessible resources [Berners-Lee et al., 2000].

The utility of ontologies can be far wider than apparently is. For instance, ontologies can be applied in E-commerce, where communication between agents can be simplified by providing a common vocabulary to describe goods and services [McGuinness, 1998]. It can also be of an hand in search-engines [Fensel et al., 2001], since it facilitates while looking for pages that contain semantically similar but syntactically different words and phrases. The final example is ontologies contribution in Web and grid service [McIlraith et al., 2001, Li and Horrocks, 2004], which can provide rich service descriptions and details that can be useful when discovering suitable services.

All these examples manage to substantiate that the creation of several ontologies implementations had as main goal to solve and provide solutions such as Resource Description Framework (RDF), Web Ontology Language (OWL), Web Service Modeling Ontology (WSMO) and Semantic Markup for Web Services (OWL-S). These technologies focus on guaranteeing that as the information is contained in documents, it needs to be processed by applications, and then ontology offers a solution.

3.5.1 Resource Descripton Framework

Resource Description Framework consists in a model for representing named properties and property values. It is capable of representing information about resources in the World Wide Web based in *XML*, as a kind of language.

The purpose of its creation was the representation of metadata about Web resources. However, over the course of time, it evolved and started to be used to represent information about identifiable things on the Internet. Nowadays, *RDF* is intended for cases where information needs to be processed by applications instead of being managed by people, providing a common framework to represent this information, which can be exchanged between applications without losing its meaning.

Its basic model consists of three object types presented in [Lassila and Swick, 1999].

These three object types are resources, properties and statements. Resources are all the things that can be described using RDF expressions. They are named by URIs and they can be any type of document: an entire web page or a simple XML element. Properties are used to describe a resource: it can be a specific aspect, attribute, characteristic or a relation. Finally, statements are resources with a named property plus the value of that property. It has three individual parts: subject, predicate and object.

3.5.2 Web Ontology Language

Despite RDF being an ontology language which can formally describe the meaning of terminology used in Web documents, if machines were expecting to perform useful reasoning tasks on these documents, RDF would not be enough to satisfy these needs. In this matter, Web Ontology Language (OWL) can bring more details on ontologies enabling a higher machine understanding between them.

OWL can offer three types of sublanguages that can fit consonant user needs [McGuinness et al., 2004]: OWL Lite for the users that mainly need a classification hierarchy and simple constraints, OWL DL known for having the maximum expressiveness keeping computational completeness and decidability and OWL Full which supports all the options available such as the maximum expressiveness.

Despite having these three different types of sublanguages, OWL has the structure of the properties and classes of RDF schema and the basic "fact-stating ability of RDF", exploring and extending them in important ways. In OWL the user can define classes with a single property where all the values of that property in instances of the class must belong to a certain class. [Horrocks et al., 2003] provided an example to show and explain what OWL can add from RDF: working with RDF it is possible for users to declare and use classes like Country, Person, Student and state that Student is a subclass of Person and it can also state that Canada and England are both instances of the class Country. On the other hand, with OWL it is conceivable to state that Country and Person can be disjoint classes and state that Canada and England are distinct individuals between several possibilities.

3.6 Services on the Semantic Web

As described earlier, WSDL is one of the main languages for describing operation features of Web Services. However, WSDL does not support a semantic description of Web Services and it only includes a construction which describes from a syntactic point of view. Semantic Web Services technologies such as OWL-S are developing a way to provide semantic specifications from services. This type of information, from a semantic point of view, can enable a more flexible automation of service provision and use, or support the creation of tools and methodologies capable of interacting and extracting more information from the services [Martin et al., 2005].

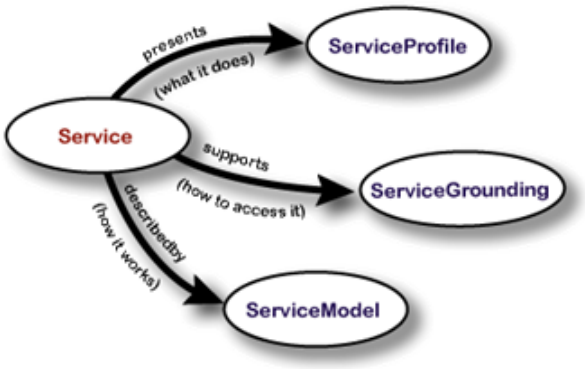


Figure 3.4: Top level of the service ontology [Martin et al., 2004]

3.6.1 Semantic Markup for Web Services

OWL-S was created as a language for describing services reflecting the fact that it uses a standard vocabulary that can be interpreted and use other aspects from OWL to create service descriptions. [Narayanan, 1999, McIlraith et al., 2001] showed the three aspects that OWL-S will enable:

- Automatic Web service discovery: An automated process that searches for web services capable of offering a specific service requested by the user. If an user wants to find a service which is capable of selling airline tickets to a specific place, with the use of ontologies based on OWL-S it is possible to find the result without human intervention.

- Automatic Web service invocation: is the automatic invocation of a Web service by a computer program when given only the description of the service chosen.

- Automatic web service composition and interoperation: this task is based on the automatic selection, composition, and interoperation of Web services to execute a task, given a description of an objective.

The structure of this ontology of service is based on three types of knowledge, presented in Fig. 3.4. The class Service is a service instance for each service found and each instance will present three distinct descriptions: ServiceProfile, ServiceGrounding and ServiceModel. The details of these descriptions may vary consonant the selected service.

The ServiceProfile ontology specifies web service descriptions based on their functional and non-functional parameters [Balzer et al., 2004]. The functional parameters are the transformation of data and states during the execution of a web service. The profile of OWL-S specifies the inputs and outputs that a web services has and the pre-conditions that must have to be able to execute the service. Non-functional-parameters are divided into two parts: semi-structured information which was created for human users to be

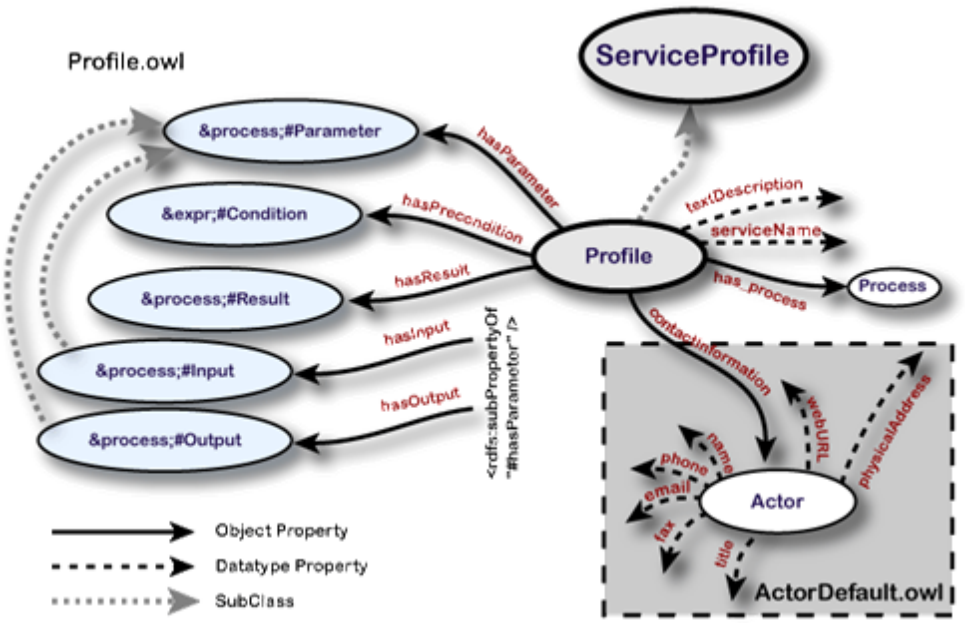


Figure 3.5: Selected classes and properties of the profile [Martin et al., 2004]

able to understand what it is in it, with no relevance for semantic service discovery (e.g. serviceName, textDescription) and sub-classes of ServiceParameter to include additional requirements that user may think that are important in his ontology.

The ServiceModel is used to provide information to the client teaching how to use the service, by detalling the semantic content of request and when the requisites and pre-requisites are filled up to receive the output of the service. Basically it will teach the user how to interact with the service and the way in which he can get the information. Zooming in, to understand how to interact with a service, it can be viewed as a process. There are three classes of processes: AtomicProcess which are descriptions of a service that expects one message and returns other message, SimpleProcess which specifies abstract views of processes by hiding fields such as inputs or outputs and CompositeProcess which maintains some sate, e.g. when a client sends a message it will pass through the processes involved in the CompositeProcess.

Finally, the ServiceGrounding specifies the details of how an agent/user can access a service. It specifies a communication protocol, messages formats and so on. In addition, ServiceGrouding for each input and output specifies a semantic type. This is used to guarantee and avoid different data elements when exchanged with the service. If a machine knows which type of serialization is used, it will communicate in an unambiguous way.

As seen in Fig. 3.5, it is possible to observe the selected properties of the subclass Profile from ServiceProfile. Each Process must have a Profile associated in order to describe it.

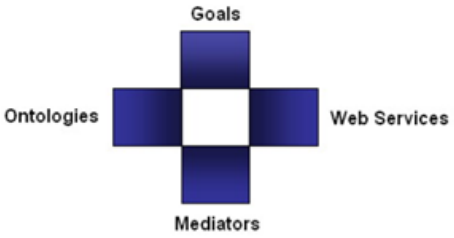


Figure 3.6: The top-level elements of WSMO [Roman et al., 2005]

3.6.2 Web Service Modeling Ontology

Web Services Modelling Ontology is also used to describe various aspects related to Semantic Web Services. It is composed by four top elements considering the main concepts: Ontologies, Web Services, Goals and Mediators (shown in Fig. 3.6).

Ontologies provide the terminology used by other WSMO elements to describe the relevant aspects of the domains of communication. Web services are the entities that provide access to the service describing the computation entity. Included in descriptions are the capabilities, interfaces and the internal working of a Web service. Goals symbolize user desires, expecting the result when executing a Web service. Mediators describe elements which are responsible for the integration between WSMO elements. They are the main responsible to allow compatibilities on the data, process and protocol level.

[Lara et al., 2005] studied the comparison between these two ontologies for services concluding that OWL-S is a more complete ontology than WSMO arguing that OWL-S is more mature in some aspects such as the definition of process model and the grounding of Web services. The main differences presented are:

1. Use of non-functional properties: WSMO defines a set of core non-functional properties which are reachable to all WSMO modelling elements while in OWL-S non-functional properties are restricted to the profile description

2. Range of non-functional properties: While WSMO recommends to use specifically vocabularies, OWL-S does not consider such specifically vocabularies

3. Description of request: While WSMO describes the request in terms of goals, OWL-S uses profiles to characterize the service being required

4. Grounding: WSMO does not have define how will ground Web Services to an invocation while OWL-S provides a grounding to WSDL as seen before

4

Architecture Overview

The present work addresses part of the test and validation of the SOA lifecycle support architecture depicted in [Cândido et al., 2011]. An architecture developed by specifying and assembling elements that can mutually interact and be combined to agile the infrastructure evolution at shop floor device level. In general terms, as shown in Fig. 4.1, the followed infrastructure defines elements and methodologies to increase devices interoperability and agility performance at device level in a service-oriented industrial automation context. Device interoperability and subsequent overall system agility can be enhanced through the employment of semantic techniques on top of a collection of devices operation in a service-oriented industrial environment able to discover, recognize and process available information to assist or automate certain integration or reengineering tasks. In this setting, the systems integrator is assisted by a distributed collection of intelligent entities and tools to complete his assignments in a more agile manner.

In a holistic overview, the application will emerge from the composition of simpler modules to progressively create more complex structures and behaviours. As presented in [Cândido et al., 2009, Cândido et al., 2010], already addressed other aspects and components of this same reference architecture. There are eleven main different elements presented in this architecture. The present work, focused on the development of three elements: *Device Explorer*, *Process Management Tools* and *Semantic Assistant*.

4.1 Device

In this context, a device, or more specifically a logical device, is seen as the main logical entity that abstracts an application element, while its hosted services represent the functionalities that it allows other to exploit. A logical device can be employed whenever there is a need to abstract a particular system component not explicitly associated to a

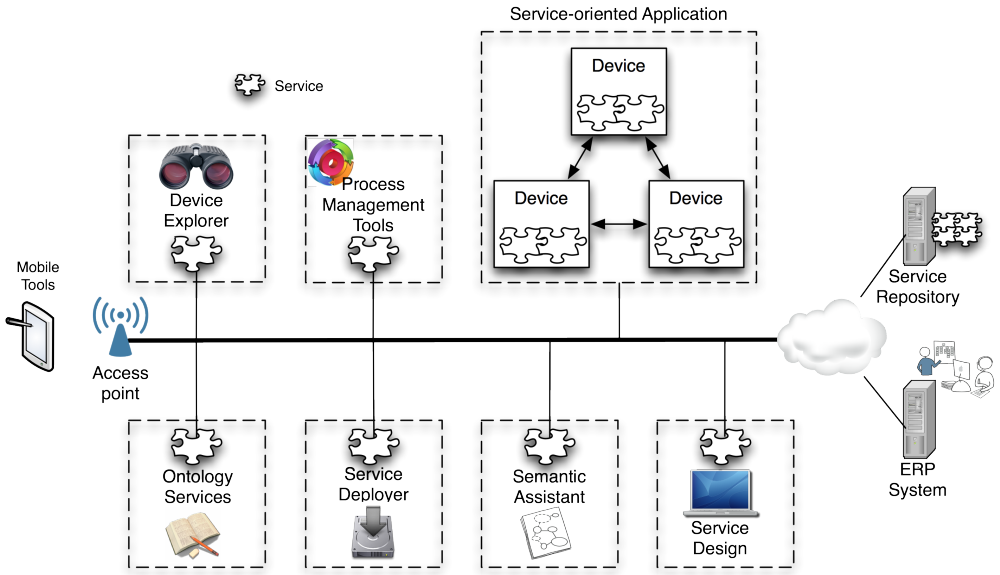


Figure 4.1: Service-oriented device lifecycle support infrastructure [Cândido et al., 2011]

physical entity. The term physical device refers to a physical system capable of interact with sensors and actuators. Consequently, a physical can be abstracted in the network as a logical device. Fig. 4.2 shows the device model which represents the device itself as a real-world physical entity, a real device.

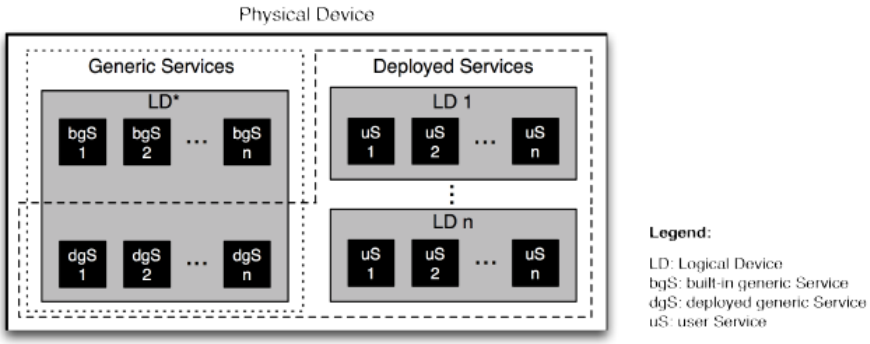


Figure 4.2: Device model[Cândido et al., 2011]

4.1.1 Web service

A web service is a software component that will encapsulate a function or behaviour under its interface following SOA principles of design and technology described in Chapter 3. It can be discovered by other network entity in order to be executed and perform a particular task or retrieve some important information. It is important to understand

that services are embedded in devices and typically offer identification, setup, monitoring and diagnosis skills.

4.1.2 Application

A service-oriented application is the result of composing several resources that cooperate between them exploiting the services each one offer in a coordinated routine. The control structure does not have a predefined standard and can be constructed based on the available resources, logical devices and services. The system integrator will have to model the interaction between these resources in order to achieve to a system that fits the desired requirements and constraints.

4.1.3 Ontology

In the scope of this work, ontology is a knowledge model that describes the overall service-oriented production system along with its components and the relations between them. As explained in 3, ontologies allow machines to interact between them with the same language not allowing any doubts about the substance of the interaction. The ontology can define which kind of services a particular range of devices expose as built-in generic services, or which services can be aggregated to support the execution of a more complex task.

4.1.4 Service Deployer

This element of the presented architecture represents a service-oriented software component that will support the deployment of the service into the devices available in environment. Inside the network this is offered as a service capable of being employed when there is a need to update device behaviour or to insert a new service, increasing the number of solutions provided by the devices.

4.1.5 Service Design

This element is not only seen as a graphical environment to construct an application based on the presented logical devices and services in the network, but also allows the design and development of customized components that can be then installed using the services responsible for it through the *Service Deployer* element.

4.1.6 Service Repository

In this infrastructure, this element is seen as a repository of logical devices and services that can be downloaded whenever they are needed. It can be updated by everyone who developed a valid resource, e.g. common control functions, particular equipment implementations, etc. and made available online. With this repository,the development of common functionalities is permitted to several domains by external users which are

allowed to download to the repository. The integrator simply needs to search over the catalogue and deploy the preferred service into the appropriated shop floor device.

4.1.7 Acess Point

The Access Point is a communication gateway for mobile devices to the existing network ecosystem. Since this architecture is only based on open web standards, interactions between this infrastructure and a mobile device are completely transparent.

In this context, access point is seen as a diagnostic tool. Since it is wireless, it enables mobility to responsible entity of the shop floor during troubleshooting, maintenance and diagnosis phases to check the status of each device or to evaluate system performance or behavior. Mobile devices such as a PDA or a tablet with wireless access can be used with a generic tool developed to manage and control the current system. Device generic built-in services will then provide basic discovery, allowing access points the capability of controling and managing the shop floor system.

4.2 *Device Explorer*

This element is a software component mostly employed for analysis, setup and troubleshooting of a service-oriented application and its constituent devices and services. It comprises a perceptive GUI that allows the integrator to discover available devices and services in the network, check their status, visualize metadata, test available services, etc. Features such as the retrieval of current devices topology and the logical view of the system based on devices and services metadata can be presented in a graphical tree map. This allows the representation of the different logical levels of an application. For instance, it permits the visualization of an orchestrator device with all its sub devices and services under it. Because of this, there is an improvement of the understanding of who controls whom, since the information about how the devices in the shop floor are structured and thought is provided to the user in a faster way. *Device Explorer* also allows the deployment of semantic translation solutions to provide a faster matching between services when executing production processes over the shop floor and the access to the *Semantic Assistant* to easily identify devices and improve posterior interactions. After discovering a device or service in the network, it will be possible to map its metadata with a serviceoriented shop floor ontology to allow a faster identification, setup and integration of that element. This will allow, for example, the retrieval of the proper set of generic services available for that particular type of device, which will then facilitate communication.

Furthermore, it allows visualization over the production processes that were executed and are being executed providing a higher knowledge about the detail of the processes.

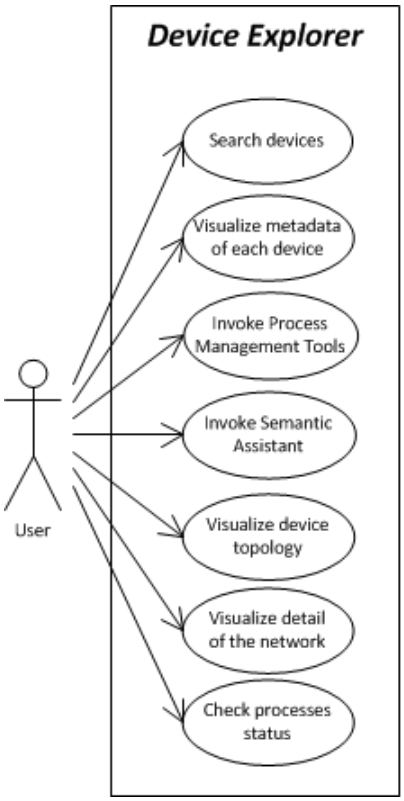


Figure 4.3: *Device Explorer* - UML use case

4.3 *Semantic Assistant*

This element, implemented by this work, does not only support the execution of reasoning tasks over the service-oriented shop floor ontology, but also exposes these functionalities as a service. These services can include the semantic identification of devices and services, retrieval of generic services for a range of devices, mapping of features and services, etc. *Semantic Assistant* is implicitly connected to the infrastructure ontology since all reasoning is based over it and conditioned by the request action input. This component can provide an important assistance to service design tools during the development of new logical devices and services by ensuring that it remains compliant with infrastructure knowledge model, system specifications and expected Quality of Service.

It presents two distinguished functionalities: *Semantic Translation* and *Semantic Gateway*. *Semantic Translation* will allow the interaction user-application and also permit the creation of matching between services. *Semantic Gateway* is also represented by a GUI, allowing user to create a service translator which will look for a semantic match to invoke a service presented in the network.

The *Semantic Assistant* can extract translation guidelines from the system ontology to determine the required translation mappings to be implemented by the gateway service. After gathering these translations guidelines it will be possible to automatically create a service possible of being deployed into a free device using the Service Deployer. This

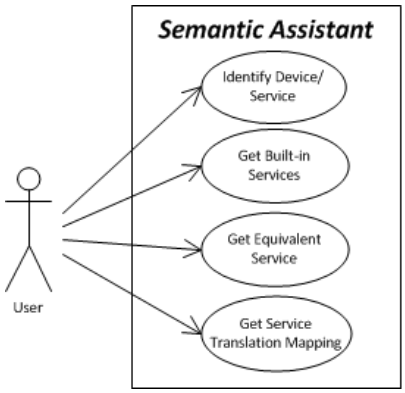


Figure 4.4: *Semantic Assistant* - UML use case

translator service will emulate, i.e. copy the interface of the missing service and translate any invocation that it receives to the new service interface that will temporarily replace the original service. For the device that was using the now inexistent service, it will discover the translator as if it were the original service and start to use it.

This *Semantic Assistant* can be configured to behave according to different levels of autonomy depending on the system critical factor set by the user. As a first step, the semantic input must be kept in the scope of systems integrator assistance. Then, regarding run-time performance and results, it can be envisaged to gradually automate certain common tasks to avoid constant need of human intervention.

4.4 *Process Management Tools*

For every industrial automation installation there is a need to define the set of processes to be executed by the system during run-time or for atypical situations. In the context of a service-oriented application, a process consists of a flow of invocations of services or events handlers provided by the available shop floor devices in a predefined way to execute predetermined productions tasks.

This element is responsible for creating and managing these processes supported by a GUI that enables a visual composition of resources and their interaction patterns to create these processes.

4.5 Wrap-up

As stated in [Cândido, 2013], the current architecture proposal introduces a set of elements to support a more agile, transparent and effortless lifecycle support to the device level in service-oriented industrial automation installations. This collection of service-oriented elements can compose a mouldable infrastructure focusing to ease common integration aspects, such as device discovery, identification, setup and process modification to cope with an unexpected event. The ability to reconfigure process plans is improved

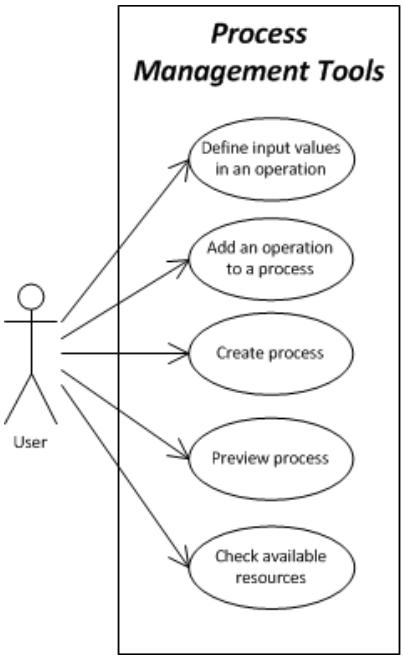


Figure 4.5: *Process Management Tools* - UML use case

when comparing with more traditional approaches in terms of interoperability and hardware abstraction targets while remaining compliant with domain know-how. By laying over open web standards and focusing on interoperability, modularity and uncomplicated management it is possible to define a set of components that together form a customizable support infrastructure. This infrastructure is modular and adaptive enough to evolve along with system specificity and requirements during its lifecycle. Each architecture element exposes its services in the network, which will enable a customized composition of modules and a mutual transparent interoperability. The ability to chose the fittest combination of elements to be set for a particular installation is a major outcome since not only it permits the customization of the end-user infrastructure but also allow equipment and solution providers to present a product offer with distinct levels of quality of service.

To sum up, the advantages against traditional control shop floor systems are this architecture's agility and adaptability whenever a problem occurs in a device. *Semantic Assistant* can provide a fast solution in order to try to replace the missing device to avoid a stop in the production system while *Process Management Tools* can redefine a process that needs to suffer small changes according to the new trends.

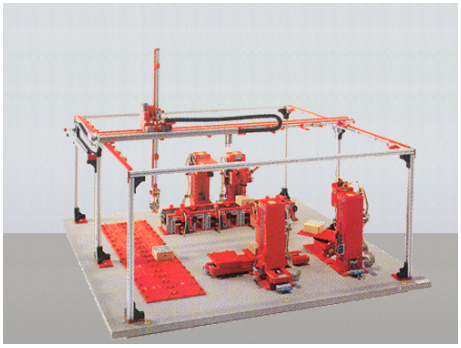
5

Implementation and Validation

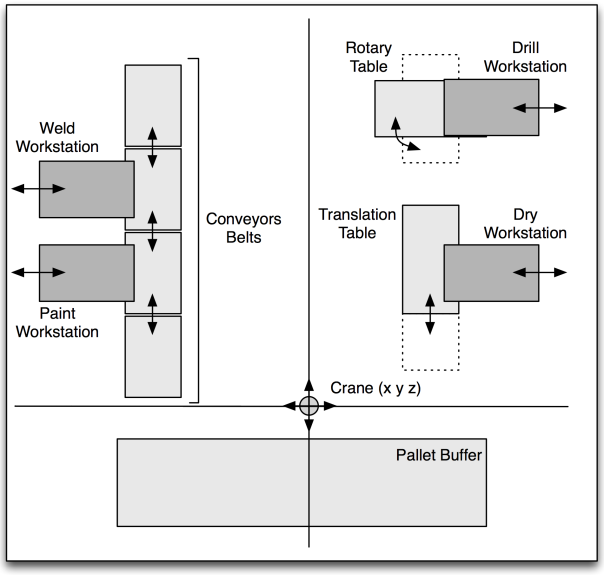
5.1 Installation

The MOFA France kit simulates a flexible manufacturing system as a closed loop manufacturing circuit, achieving various possible situations based on generic manufacturing tasks (see Fig. 5.1(a)). Staudinger models follow the modular Fischertechnik-based concept to replicate complex industrial projects in close-to-reality details. This allows systems integrators to easily discover potential problems during planning and programming, while testing suitable alternatives that can be verified quickly in a controlled environment. As verified by [Barata et al., 2008], this educational platform proved to be extremely useful for testing purposes since past experiences showed that a lot of effort can be saved if an educational platform is used and if the key aspects of a real industrial environment are taken into account, such as real-time and reliability, resources concurrency, mechanical and electrical relations, etc. Particularly relevance is the easier addition and removal of physical modules, which is much easier with this type of platform than with a real industrial system for obvious logistic reasons.

The original control equipment, composed by a legacy data acquisition board installed in a PC that allowed the acess to the kit I/O using a C++ API, was replaced by a new distributed service-oriented PLC solution composed by a distributted collection of Inico S1000 modules. The control equipment is a distributed collection of Inico S1000 modules (see Fig. 5.2). The Inico S1000 is a smart Remote Terminal Unit (RTU) device capable of real-time control, field data processing, web-based monitoring and integration with Supervisory Control And Data Acquisition (SCADA)/ Human Machine Interface (HMI) systems. It is designed to operate in typical industrial settings and is compatible with most industrial signal types and levels. The S1000 hardware configuration includes



(a) MOFA France kit



(b) MOFA France components overview

Figure 5.1: MOFA France educational kit

a 32-bit CPU running at 55 MHz and 8 MB of available flash memory, 10/100 Mbit Ethernet port, 8 digital inputs and 8 digital outputs.

This educational kit is composed of four machines that may be adaptable to different types of manufacturing tasks, a buffer area, a crane robot, local transporters (conveyors or tables) and sensors to detect pallet positions. For this particular case study, the tasks that can be performed in these machines are *Weld*, *Paint*, *Dry* and *Drill* as presented in Fig. 5.1(b). The pallets are represented by wood blocks with a carved metal ring to activate the positioning sensors. They represent pallets with product parts, subassemblies, or even raw materials that need to be transformed or processed. These pallets can be stored in the buffer area or transported by the crane to the available loading positions and then processed by a particular workstation.

Besides handling typical I/O processing, it also supports XML/SOAP interface based on DPWS to ease up the integration of industrial processes into a SOA context. However, this educational kit has some hardware limitations: it only supports two threads for input messages, to possibilitate the handling of up to two messages simultaneously, and there is only one shared thread for outgoing events and output messages so if one output message is waiting for a response, other output messages and events are waiting in a queue. The control programs can be defined using the integrated browser-based editor supporting IEC61131-3 Structured Text language and configured to be triggered whenever a linked service operation is invoked. This equipment also supports the triggering of events within control programs.

For PC-side implementations the DPWS JMEDS stack from [WS4D, 2012] and OWL-S

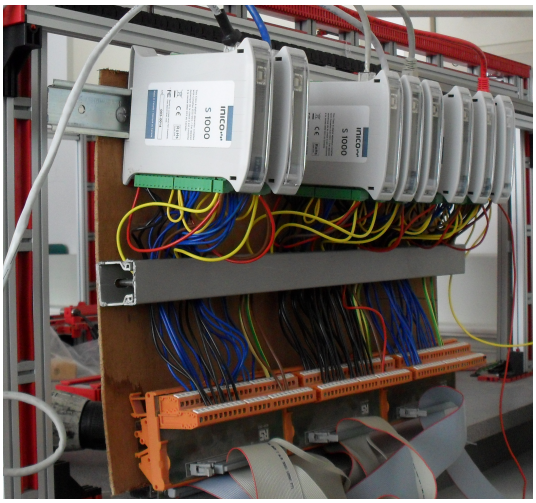


Figure 5.2: MOFA France – Inico PLC rack

API from OSIRIS Next [OSIRIS Next, 2012] were also used. DPWS JMEDS is a framework that allows the implementation and running of web services based on DPWS specification, allowing interaction with Inico devices. OWL-S API provides a JAVA API for programmatic access to create, read, write, and execute OWL-S described atomic as well as composite services.

5.2 MOFA France distributed service-oriented control overview

5.2.1 Description of Workstations

MOFA kit is composed by four workstations. After presenting it, it is relevant to explain how each workstation works. Each workstation has a machine which has positioning sensors to control and verify vertical and horizontal translations. Drill workstation is made by a machine with vertical movement and a rotary table. This table is capable of rotating 180 degrees each time to put the pallet under the machines head or to place it in a position to be picked or placed up by crane. Differently, a translation table and a set of tools are the main components of the machine which embodies the dry workstation with vertical and horizontal movement. Not only is this table capable of moving through an axis, so it can allow the reception of a pallet from two different sides and its placement over the machines head to perform an action, but it also has three types of tools to perform an operation over the pallet. Paint and Weld workstations are only made up of a machine with vertical and horizontal movement to approach and assemble the pallet under the machines heads.

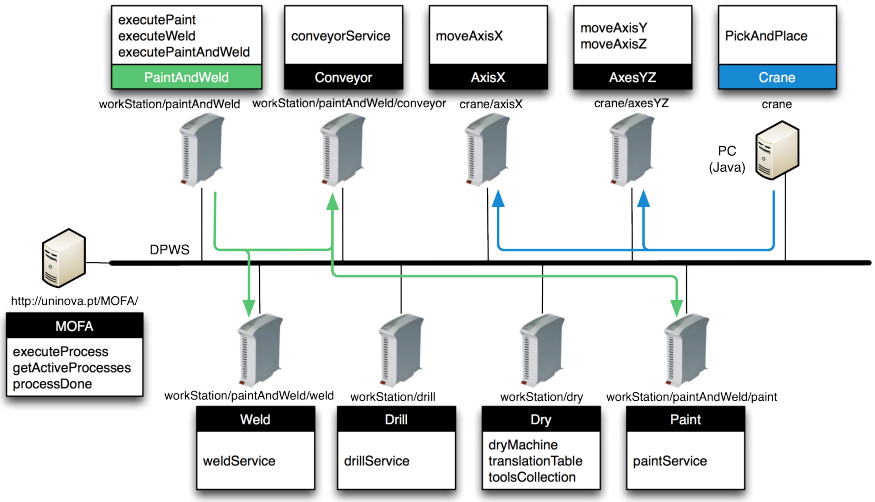


Figure 5.3: MOFA France distributed service-oriented control architecture

5.2.2 Device organization

Since devices had a limited number of I/O, 8 inputs and 8 outputs, the way in which devices would be organized in order to control MOFA kit had to be thought. Considering the number of I/O of MOFA kit, it was decided that each workstation would be controlled by one device and the same would happen to the conveyor belts: one device to control these local transporters. Meanwhile, crane robot had more I/O than other devices. Consequently, a device to control the movement in the X axis was installed as well as another device to control the two other axes: Y and Z. As a result, a virtual orchestrator, with the use of JMEDS stack, was implemented to provide a higher control over the crane robot with only one operation offered by this orchestrator to the network as it is explained in 5.2.5.

Paint and *Weld* stations with the four conveyor belts could also be gathered into another orchestrator since these three modules of the MOFA kit cannot work without the "consent" of the other two modules. By the same time, an INICO device was used to orchestrate the other three devices which control paint and weld workstations and the conveyor belts.

In this way, a complete overview of the deployed distributed service-oriented control solution for the MOFA France kit is depicted in Fig. 5.3. As this figure shows, there is a virtual device called *MOFA* which will be described in detail on the following sections as a device capable of providing services to users from other networks, executing production processes in this network and getting the status of it. It is also responsible to execute internal production processes.

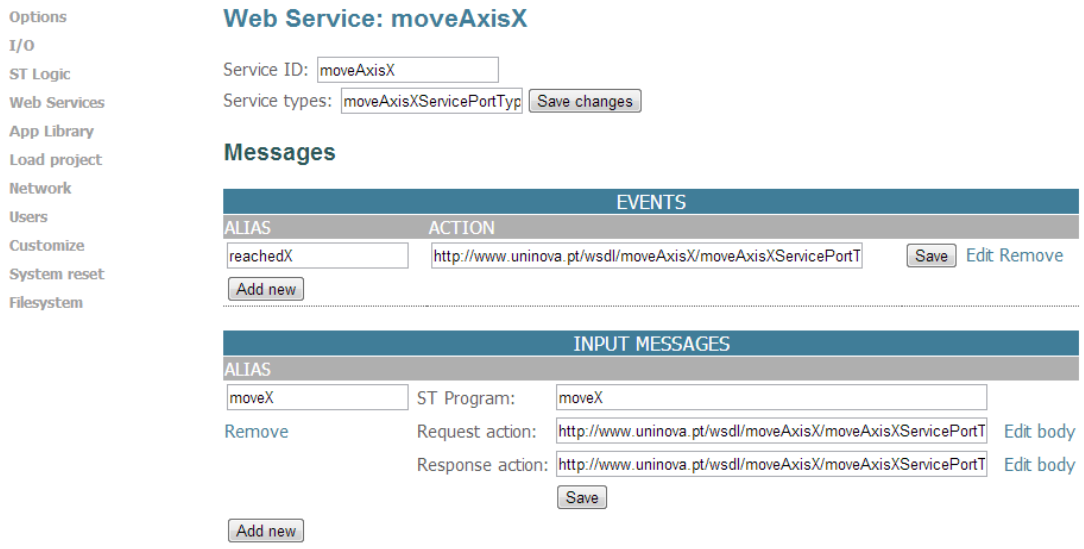


Figure 5.4: Configuration of a service in an INICO device via web browser

5.2.3 Device configuration

The S1000 programming environment allows easily defining of new Web Services, linking Web Services to ST programs, and sending/receiving messages from the logic code. To create a new Web Service in the device it is necessary to fill up some fields shown in Fig. 5.4.

The first step is to configure the field *ServiceID* and *Service Types*. *ServiceID* is a mandatory field since it is the unique identifier for the Web Service and *Service Types* is not a mandatory field since it is characterized to be a field which categorizes Web Services and is useful to organize and work with all the Web Services in a particular network. Once these two fields are entered, it is required to add messages to the Web Service.

There are three types of messages: Input, Output and Event messages. Input messages are sent from a device to the S1000. These messages will convey commands to execute a particular action or to transfer some information and it can also be used to require some information. A response message, from S1000 to device can also be configured. Output messages are sent to a device from S1000 and, like Input messages, can have or not a response message. Event messages are sent from the S1000 to one or more devices. It will report events and to receive these messages, devices must subscribe to the S1000 events so that an internal list of interested applications can be preserved.

As seen in Fig. 5.4, Input and Event Messages have a parameter called Alias used to reference a message to the ST logic program. Whenever an input message with the right request action reaches this device, it will trigger an operation called *moveY*. The request action must be identic to the incoming message. This means that it is necessary to have the message XML structure, otherwise it will not be considered to invoke this operation. The structure of the message is also defined by the user via web browser as it is shown

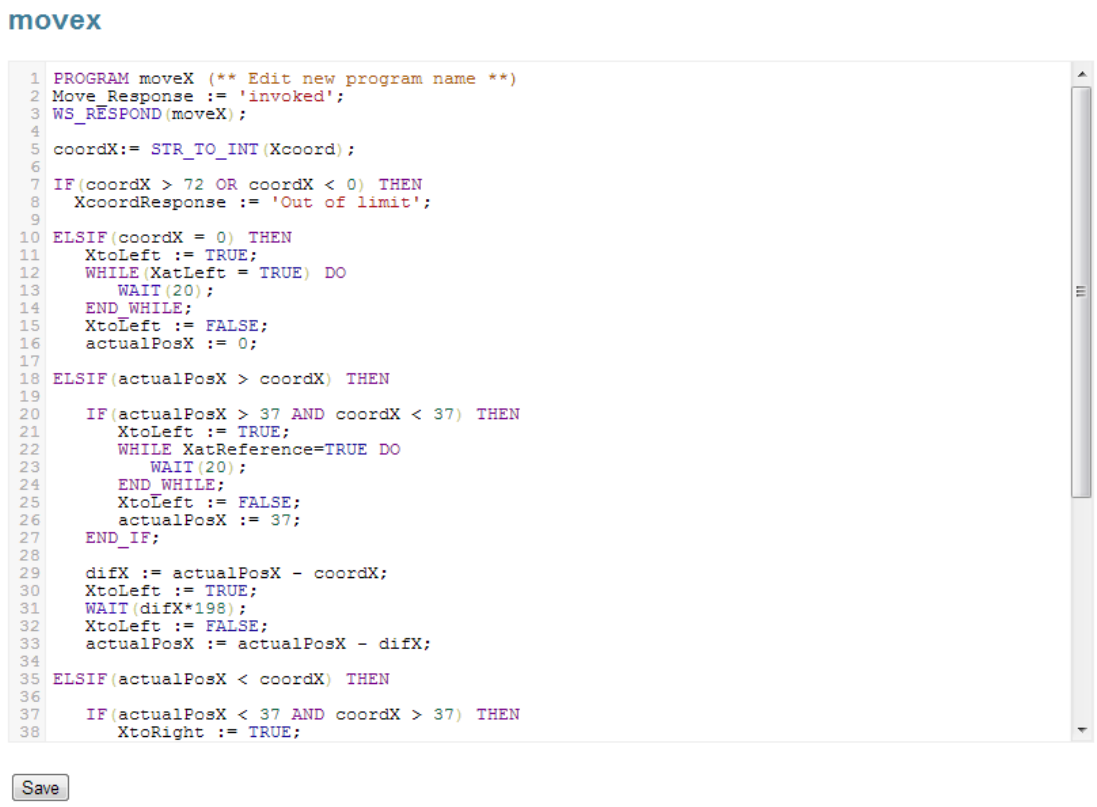


Figure 5.5: Browser-based ST programming interface for Inico S1000 devices

in the example of Listing 5.1.For the event messages, when a message is received, it will also trigger an ST logic program and will send notifications to the subscribers according to the structure of the ST logic.

Listing 5.1: Example of Input message in XML-based format

```
<moveX xmlns="http://www.uninova.pt/wsdl/moveAxisX">
  <coordX xmlns="http://www.uninova.pt/wsdl/moveAxisX">Xcoord</coordX>
</moveX>
```

Defined the service and the messages, it is also essential to program the ST logic so that when triggered by a message, it will execute an action. Despite these operations being defined using the browser-based editor supporting Structured Text (ST) language, they have some particulars functionalities: when the message is received from a PC application and the ST program is executed, it must send back a response message using the *WSRESPOND* code. When treating the ST program as part of one event, *WSPUBLISH* must be used to send automatically a message to every subscribed PC application. An example of an ST program is shown in Fig. 5.5.

Nevertheless, this configuration is not enough for a device to be discovered over the network with the use of JMEDS stack. The service must be described with the use of WSDL. For each service presented in a device a WSDL must be created so that DPWS

implementations can understand how to communicate with the devices and see which services are online. WSDL must have the same names declared in the configuration of each service, e.g. service name, etc. The WSDL capable of describing the previous service example is shown in the example of Listing 5.2.

Listing 5.2: Description of moveAxisY service in WSDL-based format

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="moveAxisX"
targetNamespace="http://www.uninova.pt/wsdl/moveAxisX"
xmlns="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:tns="http://www.uninova.pt/wsdl/moveAxisX"
xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap12/">
 <types>
  <xsd:schema targetNamespace="http://www.uninova.pt/wsdl/moveAxisX"
   elementFormDefault="qualified">
    <xsd:element name="moveXResponse">
    <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="coordXResponse" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="moveX">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="coordX" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:complexType name="reachedXType">
      <xsd:sequence>
        <xsd:element name="reach" type="xsd:string"/>
      </xsd:sequence>
  </xsd:complexType>
  <xsd:element name="reachedX" type="tns:reachedXType"/>
  </xsd:schema>
</types>
<message name="moveXRequestMsg">
  <part name="body" element="tns:moveX"/>
</message>
<message name="moveXResponseMsg">
  <part name="body" element="tns:moveXResponse"/>
</message>
<message name="reachedXMsg">
    <part name="body" element="tns:reachedX"/>
</message>
<portType name="moveAxisXServicePortType" wse:EventSource="true">
```

```
45    <operation name="moveX">
46      <input message="tns:moveXRequestMsg"/>
47      <output message="tns:moveXResponseMsg"/>
48    </operation>
49    <operation name="reachedX">
50      <output message="tns:reachedXMsg"/>
51    </operation>
52  </portType>
53  <binding name="moveAxisXServiceBinding" type="tns:moveAxisXServicePortType">
54    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
55    style="document" />
56    <operation name="moveX">
57      <soap:operation style="document" />
58      <wsdl:input>
59        <soap:body use="literal" />
60      </wsdl:input>
61      <wsdl:output>
62        <soap:body use="literal" />
63      </wsdl:output>
64    </operation>
65    <operation name="reachedX">
66      <soap:operation style="document" />
67      <wsdl:output>
68        <soap:body use="literal" />
69      </wsdl:output>
70    </operation>
71  </binding>
72  <service name="moveAxisX">
73    <port name="moveAxisXServicePort" binding="tns:moveAxisXServiceBinding">
74      <soap:address location="http://192.168.3.14:80/dpws/moveAxisX" />
75    </port>
76  </service>
77  </definitions>
```

5.2.4 Description of operations

In the present work each operation was carefully thought to provide a detailed control over the kit. Every operation interacts with sensors and motors of the kit, executing an action on it. A full list of the operations implemented to control the kit is shown in Table 5.1. Each operation is based on a request-reply communication, meaning that whenever it is invoked it will reply to the invoker with an acknowledge message when the execution reaches the end. Every input of each operation has a range of values associated, also referred in the mention table.

5.2.5 Orchestrators

Since INICO S1000 only allows a few amount of I/O, *crane* and *PaintAndWeld* devices had to be split in two or more devices. *Crane* device was divided into two devices: *axisX*,

Device Name	Service Name	Operation Name	Inputs	Description
drill	drillService	drill	drillTime [0..10000]	enables drill operation over a pallet
dry	dryMachine	dry	dryTime [0..10000]	enables dry operation over a pallet
	translationTable	switchTable	side [left,right]	translate the table to the left or right side
	toolsCollection	changeTool	toolNumer [1,2,3]	select one of the three tools do dry a pallet
weld	weldService	weld	weldTime [0..10000]	enables weld operation over a pallet
paint	paintService	paint	paintTime [0..10000]	enables paint operation over a pallet
conveyor	conveyorService	movePiece	id [1..n] action [paint,weld]	transports piece from one belt to another belt
paintAndWeld	paintAndWeld-Service	executePaint	id [1..n]	transport one pallet over the conveyor belts and paints it
	paintAndWeld-Service	executeWeld	id [1..n]	transport one pallet over the conveyor belts and welds it
	paintAndWeld-Service	executePaintAndWeld	id [1..n]	transport one pallet over the conveyor belts and paints and welds it
axisX	moveAxisX	moveX	coordX [0..72]	move crane robot to a position in the axis X
axesYZ	moveAxisY	moveY	coordY [0..67]	move crane robot to a position in the axis Y
	moveAxisZ	moveZ	coordZ [0,1]	crane robot picks or places a pallet in the current position
crane	craneService	pickPlace	coordX1 [0..72] coordY1 [0..67] coordX2 [0..72] coordY2 [0..67]	Crane robot moves to X1, Y1, pick a pallet and then moves to X2, Y2 to place it

Table 5.1: Description of the service operations

which controls the movement in the X axis, and *axesYZ*, which controls the movement in the axes Y and Z. This *crane* device was implemented using JMEDS Stack as a virtual device running on a regular PC capable of controlling the other two INICO S1000 PLCs. In the network, this new *crane* device will appear as any other device deployed on an INICO S1000 and will orchestrate both devices responsible for the axes control to enact a more complex operation such as *pickPlace*.

Once this *crane* device is started, it will execute a broadcast search in order to find *axisX* and *axesYZ* devices. Consequently, it will subscribe to three particular events: *reachedX*, *reachedY*, *reachedZ*. These events are an important part of the implementation of *pickPlace* operation offered by *crane* device because it allows the correct movement of crane. Moreover, they are intended to inform when crane reaches the expected position. Subsequently, *crane* device will be available in the network and ready to be invoked.

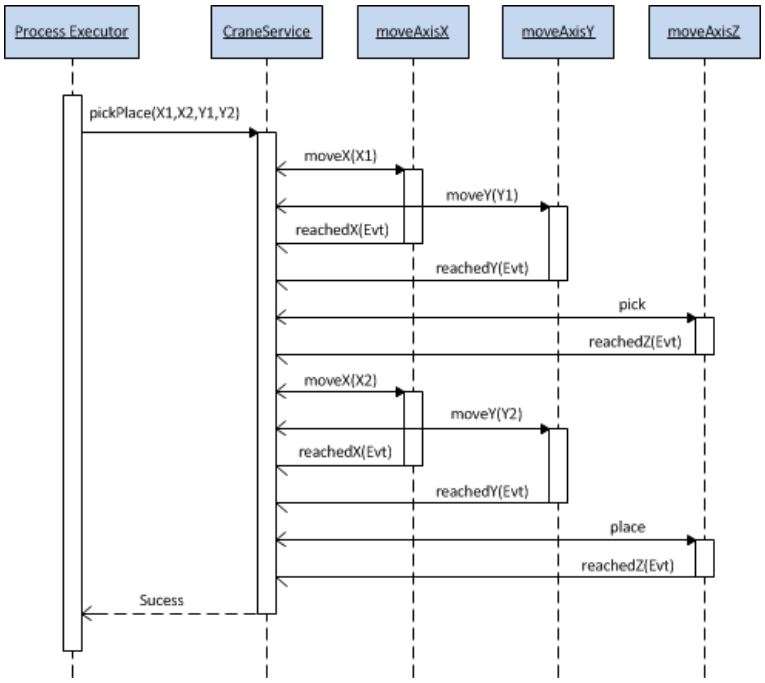


Figure 5.6: Crane Pick and Place example - UML sequence diagram

Fig. 5.6 presents the UML Sequence diagram of the messages exchanged between these three devices during a *pickPlace* operation. Each message is based on a synchronous request-reply message to indicate that the invocation was well received and the sub-operation will be executed as soon as possible. When *moveX(X1)* and *moveY(Y1)* are invoked to move the crane to a certain position, operation *pick* will not be invoked until *crane* device receives the notifications from *reachedX* and *reachedY* events. This is to make sure that crane will not try to pick or place a pallet before reaching coordinates *X1* and *Y1*. Thus, *crane* will now invoke operation *pick* and will move to *X2* and *Y2* to place the pallet following the same idea.

Having considered a virtual orchestrator, it is also reasonable to look to a S1000 orchestrator. *paintAndWeld* device orchestrates other three devices: *paint* device, *weld* device

and *conveyor* device. It supports a composed operation that executes the ampler process of welding and/or painting. The messages exchange method is very similar to the *crane* device (see Fig. 5.7).

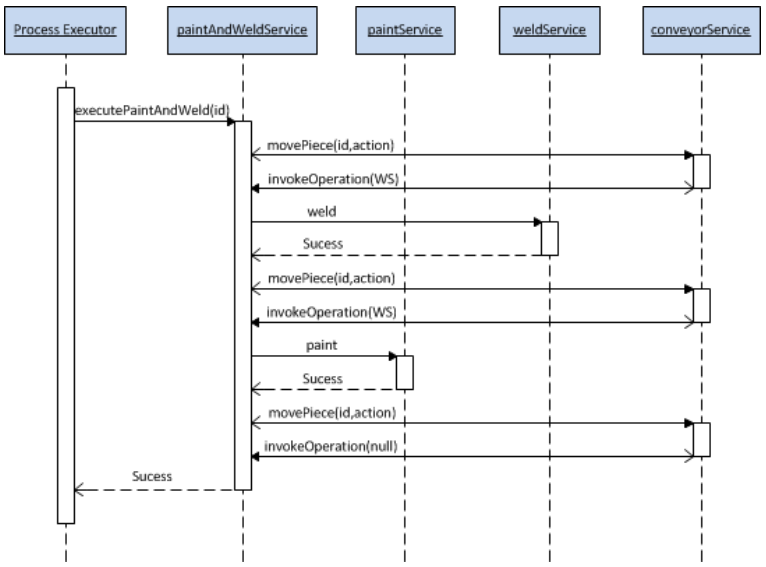


Figure 5.7: Execute Paint and Weld example - UML sequence diagram

These two orchestrators show the flexibility and versatility of this architecture. Undoubtedly that with this use of orchestrator it is possible to hide low level implementation and only focus on creating processes with high level operations.

5.3 *Device Explorer*

As referred before, this element is a software component mostly used during analysis, setup and troubleshooting of a service-oriented application. It comprises a perceptive GUI (see Fig. 5.8) that allows the integrator, by pressing *Search* button, to search the network for available devices and services, check their status, visualize metadata, or test available services.

5.3.1 Heartbeat monitoring

Even though the functioning of each device was seen as fairly reliable, a *heartbeat* event was still implemented on each one, to increase the robustness of the architecture and the time of response for anomalies that might occur. Whenever a device goes offline for some reason, e.g. connection problems, hardware conflicts or a power surge, *Device Explorer* is able to update the status of the device with the information provided by the event (or by the lack of it).

When *Device Explorer* starts to search for devices in the network, it subscribes to all events with the service named *heartbeatService*, having guarantees that each *heartbeat*

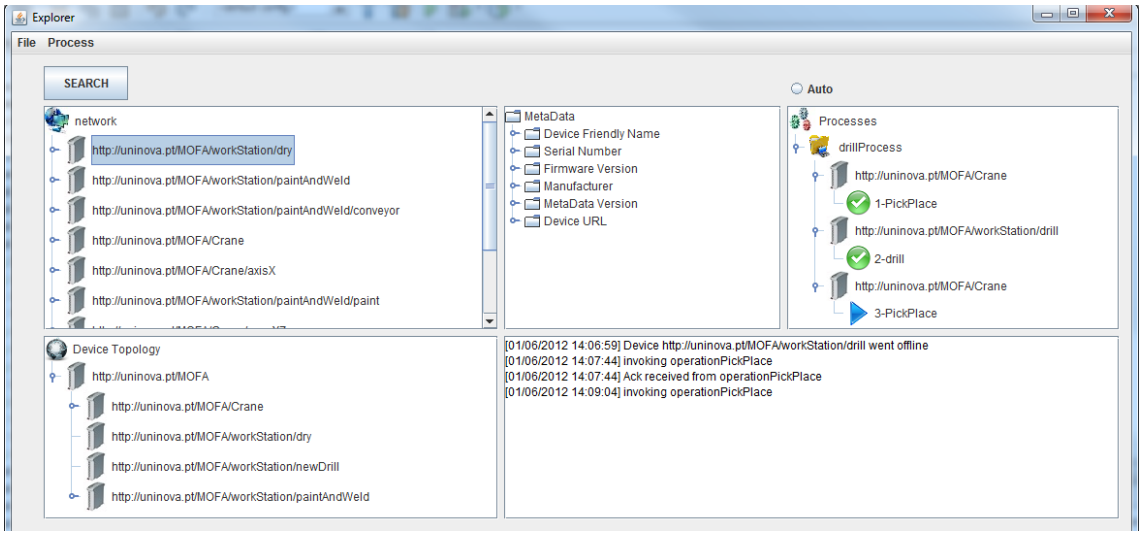


Figure 5.8: *Device Explorer* GUI

event is deployed on a service with that name. This service, through a configurable time, sends a cyclic message to all subscribers warning that it is still alive.

Hence, after subscribing to a *heartbeat* event, a control thread is associated to it and launched to monitor every change of the status of one device. This control thread activates a timer, which expires after a configurable time. This configurable time must take no longer than the time of the *heartbeat* service. However, when an event is triggered, it refreshes its own timer, resetting it to the configurable time. Consequently, when this timer expires it means that the device is no longer available in the network and *Device Explorer* will send a warning message and remove the device as seen in Fig. 5.9.

5.3.2 Granularity Variance

During the designing of operations for each device, different types of granularity in services were tested for the MOFA environment. For the *Dry* workstation, instead of creating a service capable of doing *Dry* task, several low-level services were offered, such as: *dryMachine*, *toolsCollection* and *translationTable*. This decision implies that the user has a higher level of knowledge when building production processes in order to perform *Dry* task as a whole. To execute it, there is a need to move the translation table to receive a pallet and move it again to place it under the machine. Choose the appropriate drying tool, trigger a dry operation and when it is finished, move the translation table again so that the pallet can be picked to be transported to another position by the crane.

In opposition, *Drill* workstation implements a higher-level service. *Drill* operation is the only operation invoked from the process executor to perform the whole process. When invoked, the table will rotate 180 degrees, trigger a drill operation at the drilling machine, and when it finishes, it will rotate the table again so that the pallet can be transported elsewhere by the crane.

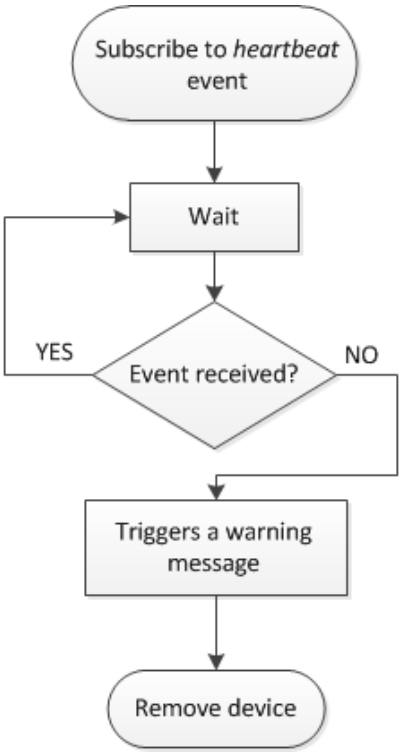


Figure 5.9: Flowchart of heartbeat control thread

Despite being two valid solutions, there are some major differences that need to be considered. *Dry* task becomes a frustrating solution when composing the process plans since it requires the user to have much more knowledge of details on the dry process than expected (see Fig. 5.10(a)). On the contrary, *Drill* task is based exclusively on one operation. That is to say, the user is not interested on those small details as seen in Fig. 5.10(b).

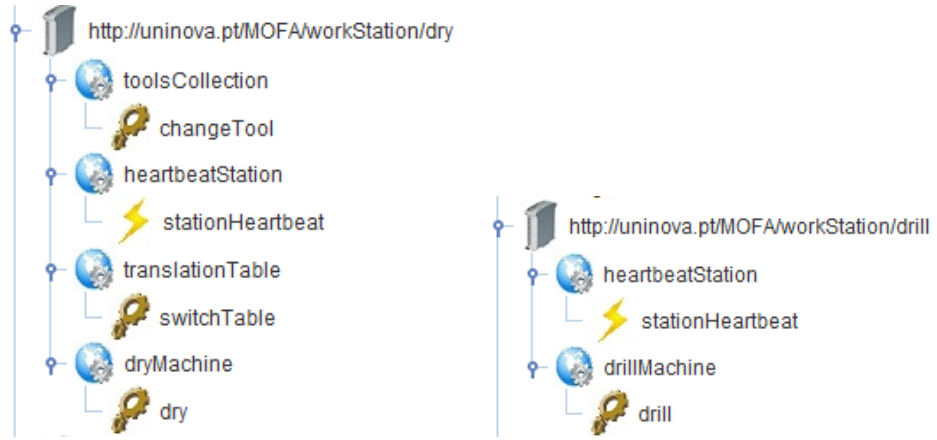
For this scenario, the former analysis suggests that *Drill* operation is a better solution. The user wants to create his process plan by combining processes, which encapsulates and hides low-level operations, capable of being executed to the pallet instead of having to work through small details, e.g. translating table or choosing the desired tool.

5.3.3 Logical Topology

The *Device Explorer* supports a network broadcast discovery exploiting the WS-Discovery standard embedded in the DPWS stack. Not only can the *Device Explorer* discover and retrieve metadata from devices discovered in the network but it can also present the device topology. In this way the systems integrator can retrieve details about an existing device, its own metadata, hosted services, operations and event sources (see Fig. 5.11).

As an example, it is possible to observe *moveX* which is an operation and *reachedX* an event source supported by the *moveAxisX* service hosted by the device *axisX* which is responsible for the movement of the crane over the x axis.

Although Fig. 5.11 shows a full list of devices in the network, it was problematic to understand which devices were orchestrators and who controls whom. Due to this



(a) DryWorkstation detail: device, services and operations

(b) DrillWorkstation detail: device, services and operations

Figure 5.10: Different types of granularity

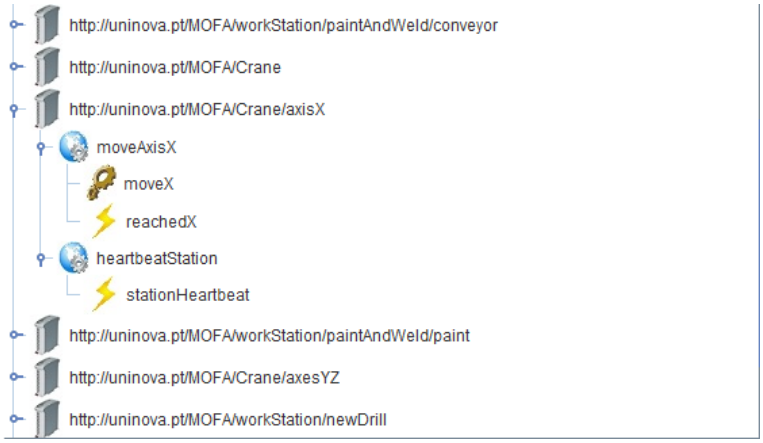


Figure 5.11: MOFA network visualization

problem, each device owns a specific *friendlyName* based on a namespace. As seen is section 5.2.5, *crane* device orchestrates two devices: *axisX* and *axesYZ*. In Fig. 5.12 it can be observed that these two devices are sub devices of the *crane* device. Their *friendlyName* is similar to a sub domain of *crane* device *friendlyName* with which the building of a device topology becomes conceivable. Each time a device is controlled by another one, its own *friendlyName* will be a subdomain of the device which controls it, e.g. it will have the same *friendlyName* of the controller device plus the name of the device. As it is shown in the figure, *crane* device orchestrates two devices, *paintAndWeld* orchestrates three devices and *MOFA* device abstracts all devices in the network as a whole shoop floor cell.

This topology shows how devices are logically composed, only based on local information exchanged and will offer an improved overview of the current system topology uniquely based on metadata held by each device to better assess the implications of modifying a part of the system or determine a fault origin.

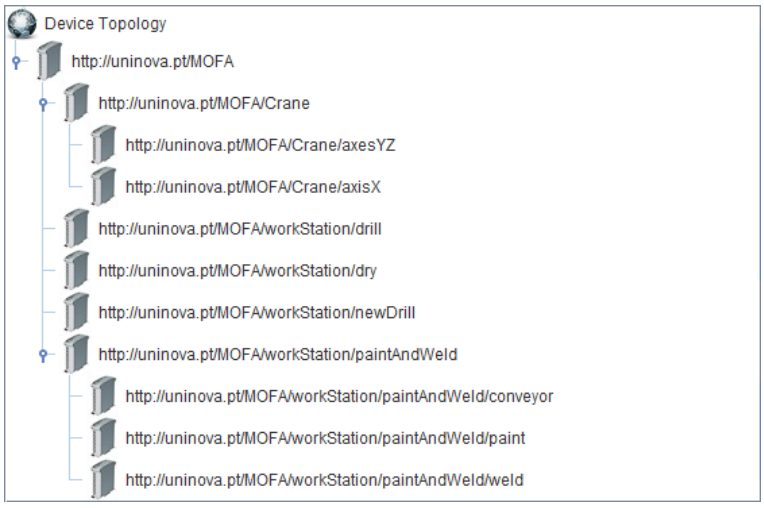


Figure 5.12: Device topology visualization

5.4 *Process Management Tools*

A service-oriented application is the result of a composition of several resources that cooperate between them exploiting the services each one offers in a coordinated routine. The control configuration is constructed based on the available building blocks – services hosted by network devices. In a holistic overview, the application will emerge from the composition of simpler modules to progressively create more complex structures and behaviors.

In this work, a simple Process Management tool was prototyped so as to allow the execution of production processes at the MOFA kit. Each process plan is composed by a list of operations from different devices needed to be performed on a particular pallet to be transformed into a production part.

As a result, a simple process management environment was created so that it could allow the execution of some production processes that use services hosted by the available devices in the network. The main goal was to make an intuitive interface to create production processes composed by different operations with different levels of granularity.

Fig. 5.13 shows the options available in the interface. The user can choose the operations directly from the list of devices discovered in the network with the help of metadata. Afterwards, he can enter the according parameters and compose them to create a sequential process plan. Each time the user adds an operation to the process, it will be available to a process tree which previews all the operations that were added previously. It also shows not only the name of operation but also the device and service names which host that operation. Inputs and their values are also available in this tree.

Once it is finished, the new process will be saved in a XML format as shown in the example of Listing 5.3 so that it can be reused in posterior executions.

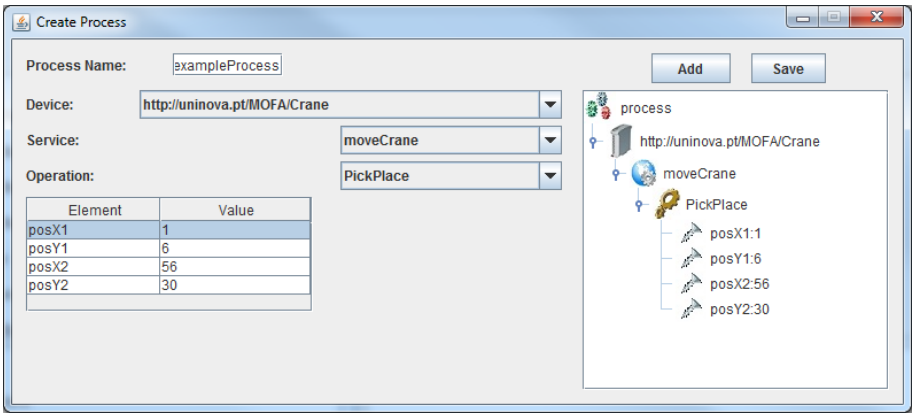


Figure 5.13: Creation of Process Plan GUI

Listing 5.3: Example of Process Plan in XML-based format

```
<xmlProcess name="drillProcess">
   <device name="http://uninova.pt/MOFA/Crane">
      <service name="craneService">
         <operation name="pickPlace">
            <input name="posX1">
               <value>1</value>
            </input>
            <input name="posY1">
               <value>6</value>
            </input>
            <input name="posX2">
               <value>56</value>
            </input>
            <input name="posY2">
               <value>30</value>
            </input>
         </operation>
      </service>
   </device>
   <device name="http://uninova.pt/MOFA/workStation/drill">
      <service name="drillService">
         <operation name="drill">
            <input name="DrillTime">
               <value>3000</value>
            </input>
         </operation>
      </service>
   </device>
</xmlProcess>
```

5.4.1 Process Executor Engine

When a systems integrator decides to launch a new process instance it can retrieve one previously stored and trigger a new generic process execution thread that will be responsible for executing and monitoring the correct invocation of each operation of the production process plan as defined in its XML file. Once a process is running, *Device Explorer* is able to show the status of each operation: if it is being executed, already finished with success or waiting to be invoked (see Fig. 5.14). This allows a better supervision over processes running at the MOFA kit.

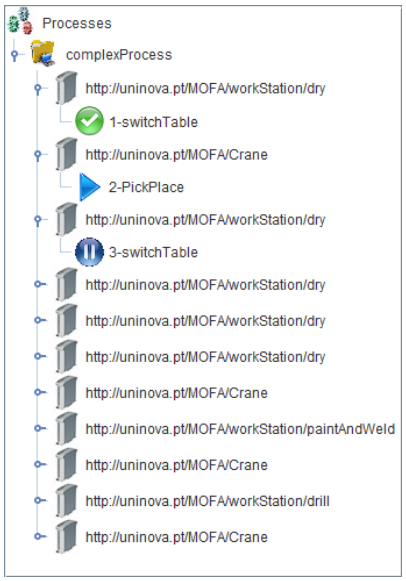


Figure 5.14: Process status

This generic process executor engine was also implemented to allow the execution of multiple processes. For this reason, each time a process is loaded, a new thread is launched and will be added to the process viewer tree. This allows a vision of all active processes.

5.5 Services Transparency

To show the transparency in each device, an INICO PLC, responsible for controlling a MOFA workstation, was disconnected, powered off, and a new one exposing similar metadata and identical service interfaces was put in place. This change showed no problems or issues: the active process ran as expected using these new services. When launching a production process, the Process Management tool searches over the network for all the required services and operations to validate if it is capable of executing successfully before launching it. If for some reason a device and consequently its hosted service become unavailable, the process task is able to handle an invocation error and start searching for an identical service in the network. If *Device Explorer* finds an identical service the

process will continue as planned.

When searching for an identical service, *Device Explorer* will execute a broadcast search looking for a service with metadata values which matches with the old service as it is shown in Fig. 5.15. This proves that IP reconfiguration does not exist, as well as no coding or any change in the process plan. It is a change done with full transparency avoiding low level details of each device: the only thing that matters is a similar metadata and service interfaces.

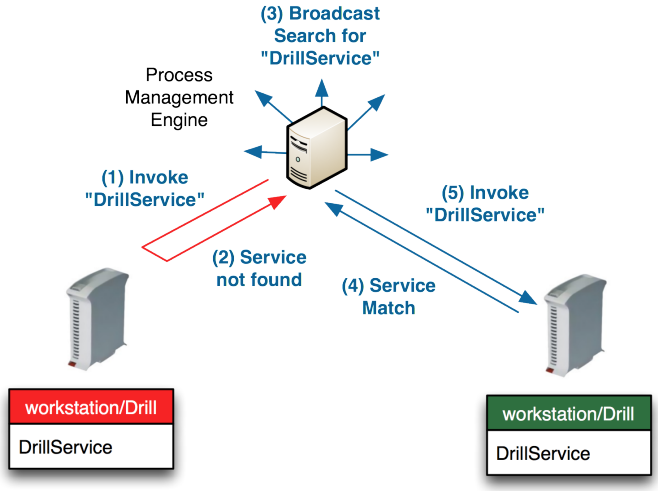


Figure 5.15: Transparent exchange of devices hosting identical services – DrillService example

5.6 Semantic Assistant

5.6.1 Semantic Matching

Adding a new device with identical metadata and hosted services of a device that was unplugged for maintenance or inactive due to fault may be a non-conceivable situation and requires the availability of an alternative solution. The present architecture can manage it when this kind of problems occur. After identifying the missing service the Semantic Assistant is requested to retrieve in the network services semantically equivalent to those which are not available at the moment. Each operation is specified through a Semantic Markup for Web Services (OWL-S) file so that each operation is semantically defined in terms of profile, functionality, parameters and grounding. Consequently, the prototyped tool is able to create automatically an OWL-S file for every existing operation by extracting information from its service description and device metadata. By having this information stored and updated, Device Explorer is able to decide which service is semantically equivalent to the missing one and proceed to the matching process.

Whenever an operation is found in the network, an OWL-S file is generated using

WSDL2OWLS tool provided by OWL-S API. This tool is used to create the basic structure of an OWL-S file with the help of device metadata and service description. Afterwards, *Device Explorer* will store an OWL-S file for each operation found and will have access to all details of device without consulting metadata or service description.

This tool is aimed to provide a translation between WSDL files and OWL-S files. Since WSDL does not provide information and details about process composition, OWL-s file will lack of information in the process field. Nevertheless, the output of this tool provides a basic structure of an OWL-S description of Web services. The translation between these two files is basically done in two main steps as explained below: every WSDL operation is equivalent to an OWL-S atomic process and XSD types are recognized as OWL-S concepts: OWL-S use concepts to specify the content of every inputs and outputs while WSDL is based on XSD types to represent the inputs and outputs of an operation.

The first step provides the basic mapping between WSDL and OWL-S. It is used to generate the fields of the basic Process Model and for generate the Grouding Model. This mapping between WSDL operation and OWL-S atomic process is realized in the following way:

- The name of each operation becomes the name of the correspondent atomic process
- The input messages of the WSDL operation become the inputs of the correspondent atomic process
- The output messages of the WSDL operation become the outputs of the correspondent atomic process

The second step is to generate the basic data into the OWL-S files (types of inputs and outputs of each WSDL operation). This conversion of types is done based on two basic rules: XSD types like string and integer are directly defined as inputs or outputs of an atomic process and the complex XSD types are interpreted as concepts whose properties correspond to the elements in the translated type.

5.6.2 Semantic Translation

Following the Semantic Matching idea, while the process engine is executing a process plan, it may not find any service to replace the missing one. Every time this occurs, *Device Explorer* will try to find a semantically similar service and provide a translation mapping. While discovery implies service description and semantic tags comparison, the invocation needs some translation mechanism to adapt to a possible unlike interface.

This approach consisted in using this mapping to invoke a service semantically equivalent when *Device Explorer* cannot find the service presented in the process plan. This decision can be done by the user, allowing him to decide which is the best option for the process, or made automatically for the use-cases that do not involve critical resources or security issues.

Despite the existence of OWL-S files there is a need of knowing how it is going to be possible to translate which service is semantically equal to another. An interface was created to help the user to build a semantic translation between two services with different interfaces (see Fig. 5.16). It shows all ontologies stored until that moment and converts them to a tree map with device, service, operation and inputs representation using *RDQL* query to read OWL-S files. By selecting one of the operations at the left one service can be mapped to another one. User defines a new service to match the service. Matching them, the system integrator in now capable of translating one service to another one despite having different interfaces. This translation is saved into a XML format as shown in example of Listing 5.4.

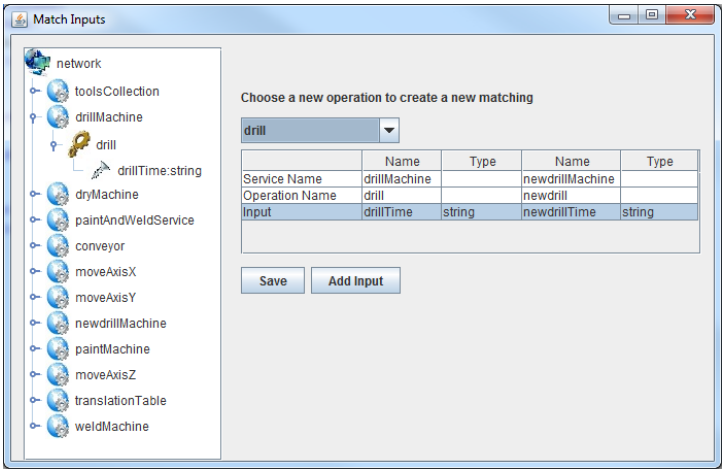


Figure 5.16: Semantic matching of services GUI

Listing 5.4: Example of Matching Services in XML-based format

```
<matching>
   <serviceMatched>
      <serviceName>drillMachine</serviceName>
      <operationName>drill</operationName>
      <serviceNameMatched>newdrillMachine</serviceNameMatched>
      <operationNameMatched>newdrill</operationNameMatched>
      <inputType>
         <name>drillTime</name>
         <nameMatched>newdrillTime</nameMatched>
         <xsdtype>string</xsdtype>
         <xsdtypeMatched>string</xsdtypeMatched>
      </inputType>
      <newInput>
         <inputName>invalid</inputName>
         <inputValue>invalid</inputValue>
      </newInput>
   </serviceMatched>
</matching>
```

In addition, user can also add new inputs without match. For instance, as seen in Fig. 5.16, *newDrillTime* is an input which matches input *drillTime*. The User may also add a new input to the new service without losing the veracity of the translation between services. This technology allows new interfaces for new operations operations to provide full versatility to devices.

The applied approach is to be executed when process executor does not find a service from a process plan. When that happens, *Device Explorer* will trigger Semantic Assistant and will look for an equivalent service, semantically equal, through the XML format file. After finding it, it will translate the old inputs to the new inputs and will add the extra inputs if necessary. If the translation of one service to another is not achievable, process engine will stop the production process, sending an error to *Device Explorer*.

In short, Semantic Translation allows a high level of matching. It enables an abstraction of low level of all details of each service when process engine is matching services.

5.6.3 Semantic Gateway

The previous use case assumed that the services were simply invoked by process engine, which is responsible for retrieving an appropriate semantic translation. However, some other network elements might be dependent of that currently inexistent service to execute their own process. In this situation the deployment of a semantic gateway will assume an essential role in this work. This gateway will copy the interface of the missing service and translate any invocation that it receives to the new interface launched in the network.

As depicted in Fig. 5.17, a semantic gateway will copy the interface of the missing device and its hosted services and translate any invocation that it receives to the semantically equivalent service interface available in the network. Regarding the device that was consuming the missing service, it will now discover the semantic gateway as if it was the original device and invoke its services again – the service interface and device metadata is the same.

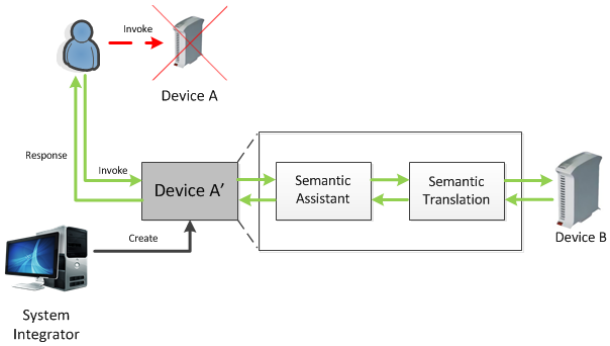


Figure 5.17: Creation of a semantic gateway example

In the current application, using the *Device Explorer* GUI, the systems integrator can

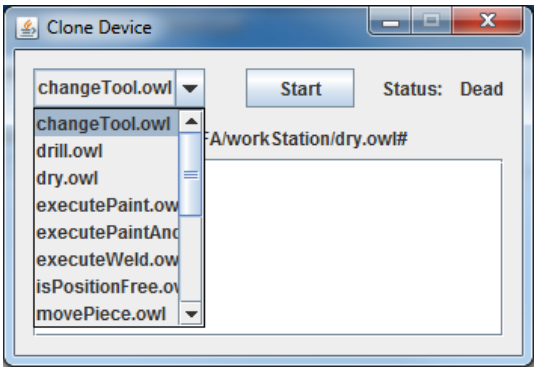


Figure 5.18: Semantic Gateway GUI

detect or be informed when a device is currently unavailable and create a semantic gateway simply by choosing the according previously stored OWL-S definition. The *Semantic Gateway* GUI (see Fig. 5.18) will then launch a DPWS device which clones the device metadata and service interfaces of the inexistent one. Internally, this new device is able to handle incoming communications (following the original missing service interface), retrieve the available semantically equivalent by exploiting the *Semantic Assistant* services and execute the translation of interface and parameters in accordance with this new semantically equivalent service. In the generic example of Fig. 5.17, device A is now unavailable, so a clone device A' is set and launched. This way the user will employ A' as if it was still using the original A, even though in reality the user is consuming the service hosted by device B.

5.7 Results

In order to test the agility of this architecture and the features of the components implemented several tests were taken into account:

1. Three XML production processes were used simultaneously.
2. A device was unplugged of the network to test the semantic matching between services.
3. A virtual device was launched to replace a device that no longer exists.

In order to test the agility of this architecture and the features of the components implemented, three processes XML-based were used simultaneously. When launching *Device Explorer* the first step to follow is clicking in the emphStart button. This will trigger the search over the network for devices and will subscribe to all *heartbeat* events to monitoring devices.

To validate the elements implemented in this architecture, three production processes were launched by *Device Explorer*. The first process consists in a drill operation over a

pallet by transporting it from the warehouse and returning it after drilled. The second one involves a dry operation over the pallet. Since dry workstation is composed by low-level and detailed actions as referred before, a series of operations must be invoked to complete a dry process. This process is composed also by a pick and place operation to the dry workstation and when finished, returns to the warehouse. The third production process loaded in parallel is a more complex one. It involves an interaction with all workstations: in first step the pallet will be taken from the warehouse and will be left in the dry station. Afterwards, it will be taken to the conveyor belt where it will be painted and will be picked by the crane robot at the exit of the transporter and moved to the dry station where it will be drilled and transported to the warehouse once more.

Before executing these three processes, the device responsible for hosting *drillService* was unplugged from the network to test the device exchange agility in this work. In the network, there is a device also capable of providing a drill service called *newDrillService* but is not referred in any xml-based production process as the service capable of drilling (*drillService*) is the one referred in the production processes files.

When executing these three processes in parallel, *process executor engine* will try to transport the pallet to the chosen workstation in the XML file. Since there is only one crane robot capable of transporting pallets it is impossible to respond immediately to them. Pick and place invocations will stay in a queue waiting for their turn in order to move the pallet to the workstation. One of the processes is based only on a drill service and when *process executor engine* tries to invoke it, finds out that this service is missing (unplugged from the network previously). Since this architecture has as a goal to enhance the device exchange agility, when the service was not found, *Semantic Assistant* was called to search an equivalent service, a semantic match, to provide to the process in order to proceed with the next actions. *newDrillService* was the result found by *Service Matching* providing it to the process executor engine to replace the missing one. While this matching was happening the other processes ran without any communication problems between devices.

Similarly, in the complex process, the pallet starts by leaving the warehouse. It then reaches the dry workstation where a dry is completely made. The pallet is then taken by the crane robot to the conveyor belt in order to be moved by it to the paint workstation where it is finally painted. Then again, the conveyor moves along the station to bring the pallet to its exit. At the same time the crane gets there and takes the pallet to the drill station. At this point, as mentioned, the device responsible for hosting drillService is unplugged from the network. Because of this, the Semantic Assistant had to be, once again, of assistance, searching, finding and putting to use the newDrillService, which is the equivalent matched service.

Testing another feature of this work, *Semantic Gateway* was invoked to create and launch a virtual device in the network with the same metadata and operations of the *drill service* (which was unplugged again). With this approach, other pc applications from other networks, had the possibility to run previously stored processes plans that included

now inexistent services but that were still available through these semantic gateways. From the point of view of the process plan execution, all required devices and service were available and ready to be used. Although this particular solution involves an extra communications cost both in terms of bandwidth and round-trip delay plus some other security concerns, it proved to ensure an immediate solution while a more definite one is being set.

These tests were performed in the MOFA kit, which was connected to the devices during twelve hours and communicating with this architecture without any problem or error of communication. Despite the tests being executed in this particular kit, this architecture proved to be generic enough to work and interact with other systems. The communication with devices is completely independent of the platform in which they are inserted.

After performing several try outs, results showed that the tasks of discovering and identifying new devices, as well as providing assistance when a device is down or disconnected gave a serious contribution: *Service Matching* can avoid stopping a whole production system by guaranteeing that if there is a device capable of performing the tasks of an offline device, this new device can replace it. Also, *Semantic Gateway* can save programming effort by providing a virtual device, responding to external needs of other networks which are not updated to the new devices presented in the system. The agility of the overall system can actually be increased, while mending with operation disruptions or modifications at device level.

6

Conclusions and Future Work

This chapter completes this document, summarizing the work done in this thesis. It also proposes future work in this area to extend the capabilities of the technology developed.

6.1 Conclusions

The current infrastructure presents an innovative solution to ease reengineering interventions in a service-oriented industrial automation scenario. This work validates the feasibility and aptness of the approach previously presented in [Cândido et al., 2011] in an educational shop floor cell test bench installation that can definitely be generalized to a bigger range of industrial automation deployments.

By offering a replacement proposal for an inexistent or faulty device whenever there is a need to replace it, along with the ability of launching a clone device that emulates the replaced device and conveys its messages to its semantically equivalent, this approach promises to deliver a relevant impact to ease system integrator role along system lifecycle. The performed tests revealed that the tasks of discovering and identifying new devices, as well as providing assistance when a device is down or disconnected offered a valuable contribution and it can increase the agility of the overall system when dealing with operation disruptions or modifications at device level.

In a more particularly case, the example of powering off *drillService* and powering on *newDrillService* to check if the system would recognize a device capable of replacing the actions performed of the unplugged device, proved that this architecture can be flexible enough to search for new control solutions that might exist in the network without the human intervention and without stopping the production system if a solution is found.

Since it is supported by open web standards, this infrastructure leaves the door open for more Interned-based applications. Although most of the overall infrastructure has

been already deployed and validated, more extensive validation is still required to fully assess the fitness of this approach to enable a more agile environment in terms of real-time performance, communications reliability and security of access to devices.

To sum up, this infrastructure proved to be modular and adaptive enough to evolve along with the system specificity and requirements during its lifecycle. Since each infrastructure element exposes its services in the network in an independent way, it is possible to introduce new elements to this infrastructure allowing and increasing the functionalities of it acoording to industry needs.

6.2 Future Work

When developing *Process Management Tools*, the focus was not put concretely on providing an advanced process management environment but on allowing the creation of some simple processes that employed services hosted by the available devices. Future work includes the use of more sophisticated service-oriented process execution solutions such as Web Services Business Process Execution Language (WS-BPEL)[Andrews et al., 2003]. A tool based on drag and drop services to make an intuitive and fast instrument capable of design production processes based on services presented or not in the network.

Also, as a plus to this work, a creation of a schedule system in the *process executor engine* would fit in this implementation to avoid collisions when two processes want the same action in a production line. A system capable of scheduling parts of the process or of finding a semantic match in other workstation when the chosen one is occupied. This would provide a higher efficacy and use of the production line, trying to produce every pallet in a free station capable of it. This could be the next step in this work, not only enhancing device exchange agility but also trying to understand which workstation is semantically equivalent to avoid waiting for the chosen workstation.

Even though this work is supported by open web standards, an effort on creating an ontology to describe services and at the same time, in some way, use one field of this ontology to machines understand which services are semantic equivalent without the use of any XML file. As a service is found in the network, an ontology could be capable of describing it semantically and understand automatically which service can replace. Despite being a valid solution, it would involve a step out of the idea of using open web standards for now. A complex ontology would be necessary as well as a simple system capable of finding solutions when a service is not found.

Despite being tested in MOFA kit and proving to be completely independent from the system controlled by it, it would be interesting for part of this architecture to see its concepts of agility and flexibility being tested in a real shop floor system.

6.3 Scientific Contributions

This work resulted in one scientific contribution that have been published in:

1. Cândido, G., Sousa, C., Di Orio, G., Barata J., and Colombo A. Enhancing device exchange agility in Service-oriented industrial automation. Proceedings of the 22nd IEEE International Symposium on Industrial Electronics, ISIE 2013

Bibliography

[Andrews et al., 2003] Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F., Liu, K., Roller, D., Smith, D., Thatte, S., et al. (2003). Business process execution language for web services.

[Atkinson et al., 2002] Atkinson, B., Della-Libera, G., Hada, S., Hondo, M., Hallam-Baker, P., Klein, J., LaMacchia, B., Leach, P., Manferdelli, J., Maruyama, H., et al. (2002). Web services security (ws-security). *Public Draft*.

[Babiceanu and Chen, 2006] Babiceanu, R. and Chen, F. (2006). Development and applications of holonic manufacturing systems: a survey. *Journal of Intelligent Manufacturing*, 17(1):111–131.

[Bajaj et al., 2006] Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Nadalin, A., et al. (2006). Web services policy 1.2-framework (ws-policy). *W3C Member Submission*, 25:12.

[Ballinger et al., 2004] Ballinger, K., Box, D., Curbera, F., Davanum, S., Ferguson, D., Graham, S., Liu, C. K., Leymann, F., Lovering, B., Nadalin, A., et al. (2004). Web services metadata exchange (ws-metadataexchange). *OASIS draft, September*.

[Balzer et al., 2004] Balzer, S., Liebig, T., and Wagner, M. (2004). Pitfalls of owl-s: a practical semantic web use case. In *Proceedings of the 2nd international conference on Service oriented computing*, pages 289–298. ACM.

[Barata and Camarinha-Matos, 2000] Barata, J. and Camarinha-Matos, L. (2000). Shop floor reengineering to support agility in virtual enterprise environments. *E-Business and Virtual Enterprises*, pages 381–394.

[Barata et al., 2008] Barata, J., Camarinha-Matos, L., and Cândido, G. (2008). A multiagent-based control system applied to an educational shop floor. *Robotics and Computer-Integrated Manufacturing*, 24(5):597–605.

[Barata et al., 2001] Barata, J., Camarinha-Matos, L., Leitão, P., Boissier, R., Restivo, F., and Raddadi, M. (2001). Integrated and distributed manufacturing, a multi-agent perspective.

[Barata et al., 2007] Barata, J., Onori, M., Frei, R., and Leitão, P. (2007). Evolvable production systems in a rms context: enabling concepts and technologies.

[Bellwood et al., 2002] Bellwood, T., Clément, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., Januszewski, K., Lee, S., McKee, B., Munter, J., et al. (2002). Uddi version 3.0. *Published specification, Oasis*, 5:16–18.

[Berners-Lee et al., 2000] Berners-Lee, T., Fischetti, M., and Foreword By-Dertouzos, M. L. (2000). *Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor*. HarperInformation.

[Bohn et al., 2006] Bohn, H., Bobek, A., and Golatowski, F. (2006). Sirena-service infrastructure for real-time embedded networked devices: A service oriented framework for different domains. In *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, pages 43–43. IEEE.

[Box et al., 2000] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S., and Winer, D. (2000). Simple object access protocol (soap) 1.1.

[Camarinha-Matos and Barata, 2002] Camarinha-Matos, L. and Barata, J. (2002). Contract-based approach for shop-floor re-engineering.

[Camarinha-Matos and Vieira, 1999] Camarinha-Matos, L. M. and Vieira, W. (1999). Intelligent mobile agents in elderly care. *Robotics and Autonomous Systems*, 27(1):59–75.

[Cândido et al., 2011] Cândido, G., Colombo, A., Barata, J., and Jammes, F. (2011). Service-oriented infrastructure to support the deployment of evolvable production systems. *Industrial Informatics, IEEE Transactions on*, 7(4):759–767.

[Cândido et al., 2009] Cândido, G., Jammes, F., Barata, J., and Colombo, A. (2009). Generic management services for dpws-enabled devices. In *Industrial Electronics, 2009. IECON'09. 35th Annual Conference of IEEE*, pages 3931–3936. IEEE.

[Cândido et al., 2010] Cândido, G., Jammes, F., de Oliveira, J., and Colombo, A. (2010). Soa at device level in the industrial domain: Assessment of opc ua and dpws specifications. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 598–603. IEEE.

[Cândido, 2013] Cândido, G. M. (2013). Service-oriented architecture for device lifecycle support in industrial automation.

[Cannata et al., 2008] Cannata, A., Gerosa, M., and Taisch, M. (2008). Socrades: A framework for developing intelligent systems in manufacturing. In *Industrial Engineering and Engineering Management, 2008. IEEM 2008. IEEE International Conference on*, pages 1904–1908. IEEE.

[Chandrasekaran et al., 1999] Chandrasekaran, B., Josephson, J. R., and Benjamins, V. R. (1999). What are ontologies, and why do we need them? *Intelligent Systems and Their Applications, IEEE*, 14(1):20–26.

[Channabasavaiah et al., 2003] Channabasavaiah, K., Holley, K., and Tuggle, E. (2003). Migrating to a service-oriented architecture. *IBM DeveloperWorks*, 16.

[Christensen et al., 2001] Christensen, E., Curbera, F., Meredith, G., Weerawarana, S., et al. (2001). Web services description language (wsdl) 1.1.

[Cucinotta et al., 2009] Cucinotta, T., Mancina, A., Anastasi, G. F., Lipari, G., Mangeruca, L., Checcozzo, R., and Rusinà, F. (2009). A real-time service-oriented architecture for industrial automation. *Industrial Informatics, IEEE Transactions on*, 5(3):267–277.

[Curbera et al., 2002] Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., and Weerawarana, S. (2002). Unraveling the web services web: an introduction to soap, wsdl, and uddi. *Internet Computing, IEEE*, 6(2):86–93.

[Curbera et al., 2004] Curbera, F., Ferguson, D., Graham, S., Niblett, P., Saiyed, J., Smith, B., and Weerawarana, S. (2004). Web services eventing (ws-eventing).

[de Deugd et al., 2006] de Deugd, S., Carroll, R., Kelly, K. E., Millett, B., and Ricker, J. (2006). Soda: Service oriented device architecture. *Pervasive Computing, IEEE*, 5(3):94–96.

[De Masi, 1999] De Masi, D. (1999). *A sociedade pós-industrial*. Senac.

[Fensel et al., 2001] Fensel, D., McGuiness, D., Schulten, E., Ng, W. K., Lim, G. P., and Yan, G. (2001). Ontologies and electronic commerce. *Intelligent Systems, IEEE*, 16(1):8–14.

[Ford and Crowther, 1988] Ford, H. and Crowther, S. (1988). Today and tomorrow.

[Frei et al., 2007] Frei, R., Barata, J., and Onori, M. (2007). Evolvable production systems context and implications. In *Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on*, pages 3233–3238. IEEE.

[Goldman et al., 1995] Goldman, S., Nagel, R., and Preiss, K. (1995). Agile competitors and virtual organizations.

[Groover, 2007] Groover, M. (2007). *Automation, production systems, and computer-integrated manufacturing*. Prentice Hall Press.

[Hashimi, 2003] Hashimi, S. (2003). Service-oriented architecture explained. *ONDotnet. com (online)(geciteerd: 20 maart 2006) Beschikbaar op URL: http://www. matcom. uh. cu/Weboo/_Rainbow/Documents/ONDotNet. com_% 20Service-Oriented% 20Architecture% 20Explained. pdf*.

[Heiler, 1995] Heiler, S. (1995). Semantic interoperability. *ACM Computing Surveys (CSUR)*, 27(2):271–273.

[Horrocks et al., 2003] Horrocks, I., Patel-Schneider, P. F., and Van Harmelen, F. (2003). From shiq and rdf to owl: The making of a web ontology language. *Web semantics: science, services and agents on the World Wide Web*, 1(1):7–26.

[IEEE, 1990] IEEE (1990). Standard Glossary of Software Engineering Terminology. IEEE Computer Society – Software Engineering Technical Committee.

[Jammes et al., 2005] Jammes, F., Mensch, A., and Smit, H. (2005). Service-oriented device communications using the devices profile for web services. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*, pages 1–8. ACM.

[Jammes and Smit, 2005a] Jammes, F. and Smit, H. (2005a). Service-oriented architectures for devices-the sirena view. In *Industrial Informatics, 2005. INDIN'05. 2005 3rd IEEE International Conference on*, pages 140–147. IEEE.

[Jammes and Smit, 2005b] Jammes, F. and Smit, H. (2005b). Service-oriented paradigms in industrial automation. *Industrial Informatics, IEEE Transactions on*, 1(1):62–70.

[Jones et al., 1990] Jones, D. T., Roos, D., and Womack, J. P. (1990). *Machine that Changed the World*. SimonandSchuster. com.

[Koren, 2010] Koren, Y. (2010). *The Global Manufacturing Revolution: Product-Process-Business Integration and Reconfigurable Systems*, volume 75. Wiley.

[Koren et al., 1999] Koren, Y., Heisel, U., Jovane, F., Moriwaki, T., Pritschow, G., Ulsoy, G., and Van Brussel, H. (1999). Reconfigurable manufacturing systems. *CIRP Annals-Manufacturing Technology*, 48(2):527–540.

[Kreger et al., 2001] Kreger, H. et al. (2001). Web services conceptual architecture (wsca 1.0). *IBM Software Group*, 5:6–7.

[Lara et al., 2005] Lara, R., Polleres, A., Lausen, H., Roman, D., de Bruijn, J., and Fensel, D. (2005). A conceptual comparison between wsmo and owl-s. *WSMO Final Draft D*, 4:44.

[Lassila and Swick, 1999] Lassila, O. and Swick, R. R. (1999). Resource description framework (rdf) model and syntax specification.

[Leitão et al., 2001] Leitão, P., Barata, J., Camarinha-Matos, L., and Boissier, R. (2001). Trends in agile and cooperative manufacturing. In *Proceedings of Low Cost Automation Symposium*.

[Li and Horrocks, 2004] Li, L. and Horrocks, I. (2004). A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, 8(4):39–60.

[Marco et al., 2008] Marco, M., Leitão, P., Colombo, A. W., and Restivo, F. (2008). Service-oriented control architecture for reconfigurable production systems. In *Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on*, pages 744–749. IEEE.

[Marik and McFarlane, 2005] Marik, V. and McFarlane, D. (2005). Industrial adoption of agent-based technologies. *Intelligent Systems, IEEE*, 20(1):27–35.

[Martin et al., 2004] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al. (2004). Owl-s: Semantic markup for web services. *W3C member submission*, 22:2007–04.

[Martin et al., 2005] Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., et al. (2005). Bringing semantics to web services: The owl-s approach. In *Semantic Web Services and Web Process Composition*, pages 26–42. Springer.

[Maskell, 2001] Maskell, B. (2001). The age of agile manufacturing. *Supply Chain Management: An International Journal*, 6(1):5–11.

[McGuinness, 1998] McGuinness, D. L. (1998). Ontological issues for knowledge-enhanced search. In *Proceedings of Formal Ontology in Information Systems*, pages 302–316.

[McGuinness et al., 2004] McGuinness, D. L., Van Harmelen, F., et al. (2004). Owl web ontology language overview. *W3C recommendation*, 10(2004-03):10.

[McIlraith et al., 2001] McIlraith, S. A., Son, T. C., and Zeng, H. (2001). Semantic web services. *Intelligent Systems, IEEE*, 16(2):46–53.

[Mehrabi et al., 2000] Mehrabi, M., Ulsoy, A., and Koren, Y. (2000). Reconfigurable manufacturing systems and their enabling technologies. *International Journal of Manufacturing Technology and Management*, 1(1):114–131.

[Monostori et al., 2006] Monostori, L., Váncza, J., and Kumara, S. R. (2006). Agent-based systems for manufacturing. *CIRP Annals-Manufacturing Technology*, 55(2):697–720.

[Nagel and Dove, 1991] Nagel, R. and Dove, R. (1991). *21st century manufacturing enterprise strategy: an industry-led view*. DIANE Publishing.

[Narayanan, 1999] Narayanan, S. (1999). Reasoning about actions in narrative understanding. In *IJCAI*, volume 99, pages 350–357. Citeseer.

[Oliveira, 2003] Oliveira, J. A. B. d. (2003). Coalition based approach for shop floor agility–a multiagent approach.

[Onori et al., 2006] Onori, M., Barata, J., and Frei, R. (2006). Evolvable assembly systems basic principles. *Information Technology For Balanced Manufacturing Systems*, pages 317–328.

[OSIRIS Next, 2012] OSIRIS Next (2012). OWL-S Java API. http://on.cs.unibas.ch/owls-api/.

[Park and Ram, 2004] Park, J. and Ram, S. (2004). Information systems interoperability: What lies beneath? *ACM Transactions on Information Systems (TOIS)*, 22(4):595–632.

[Phaithoonbuathong et al., 2010] Phaithoonbuathong, P., Monfared, R., Kirkham, T., Harrison, R., and West, A. (2010). Web services-based automation for the control and monitoring of production systems. *International Journal of Computer Integrated Manufacturing*, 23(2):126–145.

[Pine and Davis, 1999] Pine, B. and Davis, S. (1999). *Mass customization: the new frontier in business competition*. Harvard Business Press.

[Piore, 1984] Piore, M. J. (1984). *The second industrial divide: possibilities for prosperity*. Basic books.

[Pohl et al., 2008] Pohl, A., Krumm, H., Holland, F., Luck, I., and Stewing, F.-J. (2008). Service-orientation and flexible service binding in distributed automation and control systems. In *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, pages 1393–1398. IEEE.

[Ribeiro et al., 2008] Ribeiro, L., Barata, J., and Mendes, P. (2008). Mas and soa: complementary automation paradigms. In *Innovation in manufacturing networks*, pages 259–268. Springer.

[Roe et al., 2005] Roe, B., Weast, J., and Yarmosh, Y. (2005). Web services dynamic discovery (ws-discovery).

[Roman et al., 2005] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., and Fensel, D. (2005). Web service modeling ontology. *Applied ontology*, 1(1):77–106.

[Shadbolt et al., 2006] Shadbolt, N., Hall, W., and Berners-Lee, T. (2006). The semantic web revisited. *Intelligent Systems, IEEE*, 21(3):96–101.

[Taylor, 2002] Taylor, F. (2002). The principles of scientific management. *The Sociology of Organizations: Classic, Contemporary, and Critical Readings*, page 24.

[Tharumarajah, 1996] Tharumarajah, A. (1996). Comparison of the bionic, fractal and holonic manufacturing system concepts. *International Journal of Computer Integrated Manufacturing*, 9(3):217–226.

[Ueda, 1992] Ueda, K. (1992). A concept for bionic manufacturing systems based on dna-type information. In *Proceedings of the IFIP TC5/WG5. 3 Eight International PROLAMAT Conference on Human Aspects in Computer Integrated Manufacturing*, pages 853–863. North-Holland Publishing Co.

[Upton, 1992] Upton, D. (1992). A flexible structure for computer-controlled manufacturing systems. *Manufacturing Review*, 5(1):58–74.

[Van Brussel et al., 1998] Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., and Peeters, P. (1998). Reference architecture for holonic manufacturing systems: Prosa. *Computers in industry*, 37(3):255–274.

[Vernadat, 1999] Vernadat, F. (1999). Research agenda for agile manufacturing. *International Journal of Agile Management Systems*, 1(1):37–40.

[Womack, 1990] Womack, J. (1990). *Machine that changed the world*. Scribner.

[Womack and Jones, 2003] Womack, J. and Jones, D. (2003). *Lean thinking: banish waste and create wealth in your corporation, revised and updated*. Free Press.

[Wooldridge, 2008] Wooldridge, M. (2008). *An introduction to multiagent systems*. Wiley. com.

[WS4D, 2012] WS4D (2012). Web Services for Devices – WS4D-JMEDS stack. http://ws4d.e-technik.uni-rostock.de/jmeds/.

[Zeeb et al., 2007] Zeeb, E., Bobek, A., Bonn, H., and Golatowski, F. (2007). Lessons learned from implementing the devices profile for web services. In *Digital EcoSystems and Technologies Conference, 2007. DEST'07. Inaugural IEEE-IES*, pages 229–232. IEEE.